SQL Server 2005 T-SQL Recipes

A Problem-Solution Approach

Joseph Sack

Apress®

SQL Server 2005 T-SQL Recipes: A Problem-Solution Approach

Copyright © 2006 by Joseph Sack

ISBN (pbk): 1-59059-570-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Tony Davis
Technical Reviewer: Evan Terry
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Julie M. Smith
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor and Artist: Kinetic Publishing Services, LLC
Proofreader: Patrick Vincent
Indexer: Brenda Miller
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 6

Full-Text Search

Full-text search functionality allows you to issue intelligent word—and phrase—searches against character and binary data, using full-text enabled operators, which can perform significantly better than a regular LIKE operator search. In this chapter, I'll present recipes that teach you how to enable full-text search capabilities in your database using Transact-SQL.

Note In SQL Server 2005, Microsoft deprecated several of the full-text system stored procedures in favor of more consistent Transact-SQL CREATE/ALTER/DROP commands. Deprecated procedures include: sp_fulltext_catalog, sp_fulltext_column, sp_fulltext_database, sp_fulltext_table, sp_help_fulltext_catalogs, sp_help_fulltext_catalogs_cursor, sp_help_fulltext_columns, sp_help_fulltext_tables, and sp_help_fulltext_table_cursor.

Full-Text Indexes and Catalogs

Full-text indexes allow you to search against unstructured textual data using more sophisticated functions and a higher level of performance than using just the LIKE operator. Unlike regular B-tree clustered or nonclustered indexes, full-text indexes are compressed index structures that are comprised of *tokens* from the indexed textual data. Tokens are words or character strings that SQL Server has identified in the indexing process. Using special full-text functions, you can extend word or phrase searches beyond the character pattern, and search based on inflection, synonyms, wildcards, and proximity to other words.

Full-text catalogs are used to contain zero or more full-text indexes, and are stored on the local hard drive of the SQL Server instance server. A full-text catalog can contain full-text indexes that index one or more tables in a single database.

SQL Server 2005 has two internal components that are used to generate and process the full-text functionality. The Microsoft Full-Text Engine for SQL Server (MSFTESQL) is used to populate the full-text indexes and catalogs and facilitate full-text searches against the database tables. The Microsoft Full-Text Engine Filter Daemon (MSFTEFD) is used to extract textual information from the document, removing non-textual data and retaining meaningful words and phrases.

SQL Server 2005 introduces a number of new Transact-SQL commands used to create, modify, and remove full-text catalog and full-text index objects. System-stored procedures used in previous versions of SQL Server have been deprecated in place of these new commands. Also new in SQL Server 2005, full-text catalogs are now backed up along with regular database backups, and thus can be restored with a database RESTORE command as well.

Creating a Full-Text Catalog

In its simplest form, you can create a new catalog just by defining its name. There are other options however, and the extended syntax for CREATE FULLTEXT CATALOG is as follows:

```
CREATE FULLTEXT CATALOG catalog_name
     [ON FILEGROUP 'filegroup']
     [IN PATH 'rootpath']
     [WITH ACCENT_SENSITIVITY = {ON|OFF}]
     [AS DEFAULT]
     [AUTHORIZATION owner_name ]
```

The arguments of this command are described in Table 6-1.

Table 6-1. *CREATE FULLTEXT CATALOG Arguments*

Argument	Description
catalog_name	The name of the new full-text catalog.
filegroup	Designates that the catalog will be placed on a specific filegroup. If this isn't designated, the default filegroup for the database is used.
rootpath	Allows you to specify a non-default root directory for the catalog. For example in this recipe, the new catalog will be created by default on the local C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\FTData\cat_Production_Document directory.
ACCENT_SENSITIVITY = {ON|OFF}	This option allows you to choose whether the indexes will be created within the catalog as accent sensitive or accent insensitive. Accent sensitivity defines whether or not SQL Server will distinguish between accented and unaccented characters.
AS DEFAULT	This option sets the catalog as the default catalog for all full-text indexes which are created in the database without explicitly defining an owning full-text catalog.
owner_name	The AUTHORIZATION option determines the owner of the new full-text catalog, allowing you to choose either a database user or a role.

In this first example, a new full-text catalog is created in the AdventureWorks database (note that a full-text catalog only belongs to a *single* database):

```
USE AdventureWorks
GO
CREATE FULLTEXT CATALOG cat_Production_Document
```

In the second example, a new full-text catalog is created designating a non-default file path, and with accent sensitivity enabled:

```
USE AdventureWorks
GO
CREATE FULLTEXT CATALOG cat_Production_Document_EX2
IN PATH 'C:\Apress\Recipes\FTC'
WITH ACCENT_SENSITIVITY = ON
```

How It Works

In this recipe, I demonstrated how to create a new full-text catalog using the `CREATE FULLTEXT CATALOG` command. This command creates related files on the local server of the SQL Server instance. Once it's created, you'll see a new folder created with the name of the new full-text catalog. This folder will contain a subfolder and system files, none of which should be opened or modified outside of SQL Server.

Once a full-text catalog is created, you can then proceed with full-text indexes, which are reviewed in the next recipe.

Creating a Full-Text Index

In this recipe, I'll demonstrate how to create a full-text index on columns in a table, so that you can then take advantage of the more sophisticated search capabilities shown later on in the chapter.

The command for creating a full-text index is `CREATE FULLTEXT INDEX`. The syntax is as follows:

```
CREATE FULLTEXT INDEX ON table_name
[(column_name [TYPE COLUMN type_column_name]
[LANGUAGE language_term] [,...n])]
KEY INDEX index_name
[ON fulltext_catalog_name]
[WITH
{CHANGE_TRACKING {MANUAL | AUTO | OFF [, NO POPULATION]}}
]
```

The arguments of this command are described in Table 6-2.

Table 6-2. *CREATE FULLTEXT INDEX Arguments*

Argument	Description
`table_name`	The name of the table that you are creating the full-text index on. There can only be one full-text index on a single table.
`column_name`	The listed column or columns to be indexed, which can be of the data types varchar, nvarchar, char, nchar, xml, varbinary, text, ntext, and image.
`type_column_name`	The `TYPE COLUMN` keyword token is used to designate a column in the table that tells the full-text index what type of data is held in the varbinary(max) or image data type column. SQL Server can interpret different file types, but must know exactly how to do so. In this case, the FileExtension table has ".doc" in it for each row. This tells SQL Server that the data held in Document will be of a Word Document format.
`language_term`	The optional `LANGUAGE` keyword can also be used within the column list to indicate the language of the data stored in the column. Specifying the language will help SQL Server determine how the data is parsed in the full-text indexing process and how it will be linguistically interpreted. For a list of available languages, query the sys.fulltext_languages table.
`index_name`	In order for the full-text index to be created on a table, that table must have a single-key, unique, non-nullable column. This can be, for example, a single column primary key, or a column defined with a `UNIQUE` constraint that is also non-nullable. The `KEY INDEX` clause in the `CREATE FULLTEXT INDEX` command identifies the required unique key column on the specified table.

(*Continued*)

Table 6-2. (*Continued*)

Argument	Description
`fulltext_catalog_name`	The `ON` clause designates the catalog where the full-text index will be stored. If a default catalog was identified before creation of the index, and this option isn't used, the index will be stored on the default catalog. However if no default was defined, the index creation will fail.
`CHANGE_TRACKING {MANUAL | AUTO | OFF [, NO POPULATION]}`	This argument determines how user data changes will be detected by the full-text service. Based on this configuration, indexes can be automatically updated as data is changed in the table. You also have the option of only manually repopulating the indexes at a time or on a schedule of your choosing. The `AUTO` option is designated to automatically update the full-text index as table data is modified. The `MANUAL` option means that changes will either be propagated manually by the user or initiated via a SQL Server Agent schedule. The `OFF` option means that SQL Server will not keep a list of user changes. Using `OFF` with `NO POPULATION` means that SQL Server will not populate the index after it is created. Under this option, full-text index population will only occur after someone executes `ALTER FULLTEXT INDEX`, which is reviewed in the next recipe.

In this recipe's example, a new full-text index is created on the `AdventureWorks` database's `Production.Document` table (we'll demonstrate how to query the index in future recipes). `DocumentSummary` is the column to be indexed, and `FileExtension` is the column that contains a pointer to the column's document type:

```
CREATE FULLTEXT INDEX ON Production.Document
(DocumentSummary, Document TYPE COLUMN FileExtension)
KEY INDEX PK_Document_DocumentID
ON cat_Production_Document
WITH CHANGE_TRACKING AUTO
```

How It Works

In this recipe, a new full-text index was created for the `Production.Document` table, on the `DocumentSummary` column (which has a varchar(max) data type). Note that more than one column can be designated for one full-text index. Stepping through the code, the first line designated the table the full-text index would be based on:

```
CREATE FULLTEXT INDEX ON Production.Document
```

The second line of code designated the column to be indexed, and then a pointer to the column that tells SQL Server what document type is stored in the `DocumentSummary` column:

```
(DocumentSummary, Document TYPE COLUMN FileExtension)
```

Keep in mind that the `TYPE COLUMN` clause is only necessary if you are indexing a varchar(max) or image type column, as you'll be assisting SQL Server with interpreting the stored data. Regular text data types such as char, varchar, nchar, nvarchar, text, ntext and xml don't require the `TYPE COLUMN` clause.

Next, the name of the key, non-null, unique column for the table is identified:

```
KEY INDEX PK_Document_DocumentID
```

The ON clause designates which full-text catalog the full-text index will be stored in:

```
ON cat_Production_Document
```

Lastly, the method of ongoing index population is designated for the index:

```
WITH CHANGE_TRACKING AUTO
```

Once the full-text index is created, you can begin querying it. Before we get to this however, there are other commands used for modifying or removing indexes and catalogs you should be aware of.

Modifying a Full-Text Catalog

In this recipe, I'll demonstrate ALTER FULLTEXT CATALOG, which you can use to:

- Change accent sensitive settings. Accent sensitivity defines whether or not SQL Server will distinguish between accented and unaccented characters, or treat them as equivalent characters in the search.
- Set the catalog as the default database catalog.
- REBUILD the entire catalog with all indexes in it.
- REORGANIZE the catalog, which optimizes internal index and catalog full-text structures. Microsoft calls this process a "master merge," which means that smaller indexes are physically processed (not logically, however) into one large index in order to improve performance.

The syntax for ALTER FULLTEXT CATALOG is as follows:

```
ALTER FULLTEXT CATALOG catalog_name
{ REBUILD [WITH ACCENT_SENSITIVITY = {ON|OFF} ]
     | REORGANIZE
     | AS DEFAULT
}
```

The arguments for this command are described in Table 6-3.

Table 6-3. *ALTER FULLTEXT CATALOG Arguments*

Argument	Description
REBUILD	The REBUILD option deletes the existing full-text file system files and replaces it with a new catalog (with the indexes still intact).
[WITH ACCENT_SENSITIVITY = {ON|OFF}]	The ACCENT_SENSITIVITY option can only be configured when used in conjunction with a REBUILD.
REORGANIZE	This causes SQL Server to optimize catalog structures and internal indexes, freeing up disk and memory resources for those full-text indexes based on frequently updated columns.
AS DEFAULT	Sets the catalog as the default database catalog.

In this first example in the recipe, a full-text catalog is optimized using the REORGANIZE keyword:

```
ALTER FULLTEXT CATALOG cat_Production_Document
REORGANIZE
```

In this second example, a full-text catalog is set to be the default full-text catalog for the database:

```
ALTER FULLTEXT CATALOG cat_Production_Document
AS DEFAULT
```

In this example, a full-text catalog (and all indexes within), is rebuilt along with disabling accent sensitivity:

```
ALTER FULLTEXT CATALOG cat_Production_Document
REBUILD WITH ACCENT_SENSITIVITY = OFF
```

How It Works

In this recipe, ALTER FULLTEXT CATALOG was used to optimize the indexes and internal data structures, set the catalog to the default database, and rebuild the catalog and indexes within. This command is used to maintain existing catalogs, and keep them performing at their best as data modifications are made to the underlying indexed tables.

Modifying a Full-Text Index

The ALTER FULLTEXT INDEX command can be used both to change the properties of an index and also control/initiate index population. The syntax is as follows:

```
ALTER FULLTEXT INDEX ON table_name
{     ENABLE
     | DISABLE
     | SET CHANGE_TRACKING {MANUAL|AUTO|OFF}
     | ADD (column_name
    [TYPE COLUMN type_column_name ]
    [LANGUAGE language_term] [,...n] )
    [WITH NO POPULATION]
     | DROP (column_name [,...n] )
    [WITH NO POPULATION]
     | START {FULL|INCREMENTAL|UPDATE} POPULATION
     | STOP POPULATION
}
```

The arguments of this command are described in Table 6-4.

Table 6-4. *ALTER FULLTEXT INDEX Arguments*

Argument	Description
table_name	The name of the table of the index to be modified.
ENABLE | DISABLE	The ENABLE option activates the full-text index. DISABLE deactivates a full-text index. Deactivating a full-text index means that changes to the table columns are no longer tracked and moved to the full-text index (however full-text search conditions are still allowed against the index).
SET CHANGE TRACKING {MANUAL|AUTO|OFF}	MANUAL specifies that change tracking on the source indexed data will be enabled on a schedule or manually executed basis. AUTO specifies that the full-text index is modified automatically when the indexed column(s) values are modified. OFF disabled change tracking from occurring on the-full-text index.

Argument	Description
`ADD (column_name [,...n] )`	The name of the column or columns to add to the existing full-text index.
`type_column_name`	The column used to designate the full-text index file type of the data stored in the varbinary(max) or image data type column.
`language_term`	The optional LANGUAGE keyword used within the column list to indicate the language of the data stored in the column.
`WITH NO POPULATION`	When designated, the full-text index isn't populated after the addition or removal of a table column.
`DROP (column_name [,...n] )`	The name of the column or columns to remove from the existing full-text index.
`START {FULL|INCREMENTAL|UPDATE} POPULATION`	This option initiates the population of the full-text index based on the option of `FULL`, `INCREMENTAL`, and `UPDATE`. `FULL` refreshes every row from the table into the index. `INCREMENTAL` only refreshes the index for those rows that were modified since the last population and in order for `INCREMENTAL` to be used, the indexed table requires a column with a timestamp data type. The `UPDATE` token refreshes the index for any rows that were inserted, updated, or deleted since the last index update.
`STOP POPULATION`	For very large tables, full-text index population can consume significant system resources. Because of this, you may need to stop a population process while it is in progress. For indexes created with the `MANUAL` or `OFF` change tracking setting, you can use the `STOP POPULATION` option.

In this first example, a new column is added to the existing full-text index on the `Production.Document` table:

```
ALTER FULLTEXT INDEX ON Production.Document
ADD (Title)
```

Next, a full-text index population is initiated:

```
ALTER FULLTEXT INDEX ON Production.Document
START FULL POPULATION
```

This returns a warning because the full-text index population was already underway for the table (we didn't designate the `WITH NO POPULATION` option when adding the new column to the full-text index):

```
Warning: Request to start a full-text index population on table or indexed view
'Production.Document' is ignored because a population is currently active for this
table or indexed view.
```

This next example demonstrates disabling change tracking for the table's full-text index:

```
ALTER FULLTEXT INDEX ON Production.Document
SET CHANGE_TRACKING OFF
```

This returns the following warning:

```
Warning: Request to stop change tracking has deleted all changes tracked on table or
indexed view 'Production'.
```

In this last example for the recipe, the Title column is dropped from the full-text index:

```
ALTER FULLTEXT INDEX ON Production.Document
DROP (Title)
```

How It Works

In this recipe, `ALTER FULLTEXT INDEX` was used to perform the following actions:

- Add a new column to an existing full-text index. This is useful if you wish to add additional columns to the full-text index which would benefit from more advanced searching functionality.
- Start a full-text index population (which works if the population isn't already set to automatically update). For very large tables, you may wish to manually control when the full-text index is populated, instead of allowing SQL Server to manually populate the index over time.
- Disable change tracking. This removes a log of any changes that have occurred to the indexed data.
- Drop a column from a full-text index. For example, if you have a column which isn't benefiting from the full-text index functionality, it is best to remove it in order to conserve space (from the stored indexing results) and resources (from the effort it takes SQL Server to update the data).

Other actions `ALTER FULLTEXT INDEX` can perform include disabling an enabled index using the `DISABLE` option, thus making it unavailable for us (but keeping the meta data in the system tables). You can then enable a disabled index using the `ENABLE` keyword.

Dropping a Full-Text Catalog

In this recipe, I demonstrate how to remove a full-text catalog from the database using the `DROP FULLTEXT CATALOG` command. The syntax is as follows:

```
DROP FULLTEXT CATALOG catalog_name
```

This command takes a single argument, the name of the catalog to drop. For example:

```
DROP FULLTEXT CATALOG cat_Production_Document
```

How It Works

The `DROP FULLTEXT CATALOG` references the catalog name, and doesn't require any further information to remove it from the database. If the full-text catalog was set as the `DEFAULT` catalog, you'll see the following warning:

```
Warning: The fulltext catalog 'cat_Production_Document'
is being dropped and is currently set as default.
```

Dropping a Full-Text Index

In this recipe, I'll demonstrate how to remove a full-text index from the full-text catalog using the DROP FULLTEXT INDEX command. The syntax is as follows:

```
DROP FULLTEXT INDEX ON table_name
```

This command only takes a single argument, the name of the table on which the full-text index should be dropped. For example:

```
DROP FULLTEXT INDEX ON Production.Document
```

How It Works

The DROP FULLTEXT INDEX ON command references the full-text indexed table. Since only one index is allowed on a single table, no other information is required to drop the full-text index.

Retrieving Full-Text Catalog and Index Metadata

This recipe shows you how to retrieve useful information regarding the full-text catalogs and indexes in your database by using system catalog views.

The sys.fulltext_catalogs system catalog view returns information on all full-text catalogs in the current database. For example,

```
SELECT name, path, is_default, is_accent_sensitivity_on
FROM sys.fulltext_catalogs
```

returns this:

name	path	is_ default	is_accent_ sensitivity_on
cat_Production_Document_EX2	C:\Apress\Recipes\FTC\ cat_Production_Document_EX2	0	1
cat_Production_Document	C:\Program Files\ Microsoft SQL Server\MSSQL.4\ MSSQL\FTData\ cat_Production_Document	0	1

The sys.fulltext_indexes system catalog view lists all full-text indexes in the database. For example,

```
SELECT object_name(object_id) table_name, is_active, change_tracking_state_desc
FROM sys.fulltext_indexes
```

returns this:

table_name	is_active	change_tracking_state_desc
Document	1	AUTO

The sys.fulltext_index_columns system catalog view lists all full-text indexed columns in the database. For example,

```
SELECT object_name(object_id) tblname, column_id
FROM sys.fulltext_index_columns
```

returns the table name, and the indexed columns (using the ordinal position of the column in the table):

tblname	column_id
Document	2
Document	8
Document	9

Also, the FULLTEXTCATALOGPROPERTY system function can be used to return information about a specific catalog. The syntax is as follows:

```
FULLTEXTCATALOGPROPERTY ('catalog_name' ,'property')
```

The function takes two arguments, the name of the catalog, and the name of the property to evaluate. Some of the more useful options for the property option are described in Table 6-5.

Table 6-5. *FULLTEXTCATALOGPROPERTY Property Options*

Property	Description
AccentSensitivity	Returns 1 for accent sensitive, 0 for insensitive.
IndexSize	Returns the size of the full-text catalog in megabytes.
MergeStatus	Returns 1 when a reorganization is in process, and 0 when it is not.
PopulateStatus	Returns a numeric value representing the current population status of a catalog. For example, 0 for idle, 1 for an in progress population, 2 for paused, 7 for building an index, and 8 for a full disk.

In this example, the full-text catalog population status is returned:

```
SELECT FULLTEXTCATALOGPROPERTY ('cat_Production_Document','PopulateStatus')
PopulationStatus
```

This returns "0" for idle:

```
PopulationStatus
0
```

How It Works

This recipe used three different catalog views and a system function to return information about full-text catalogs and indexes in the current database. You'll need this information in order to keep track of their existence, as well as track the current state of activity and settings.

Basic Searching

Once you've created the full-text catalog and full-text indexes, you can get down to the business of querying the data with more sophisticated Transact-SQL predicates. Predicates are used in expressions in the WHERE or HAVING clauses, or join conditions of the FROM clause. Predicates return a TRUE, FALSE, or UNKNOWN response.

Beginning with the more simple commands, the FREETEXT command is used to search unstructured text data based on inflectional, literal, or synonymous matches. It is more intelligent than using LIKE because the text data is searched by meaning and not necessarily the exact wording.

The CONTAINS predicate is used to search unstructured textual data for precise or less precise word and phrase matches. This command can also take into consideration the proximity of words to one another, allowing for weighted results.

These next two recipes will demonstrate basic searches using the FREETEXT and CONTAINS predicates.

Using FREETEXT to Search Full-Text Indexed Columns

The FREETEXT predicate is used to search full-text columns based on inflectional, literal, or synonymous matches. The syntax is as follows:

```
FREETEXT ( { column_name | (column_list) | * }
         , 'freetext_string' [ , LANGUAGE language_term ] )
```

The arguments for this predicate are described in Table 6-6.

Table 6-6. *FREETEXT Arguments*

Argument	Description
`column_name | column_list | *`	The name of the column or columns that are full-text indexed and that you wish to be searched. Designating * designates that all searchable columns are used.
`freetext_string`	The text to search for.
`language_term`	Directs SQL Server to use a specific language for performing the search, accessing thesaurus information, and removing noise words. Noise words are words that, depending on the language, cause unnecessary index bloat and do not assist with the search: for example "a" and "the."

In this example, FREETEXT is used to search data based on the *meaning* of the search term. SQL Server looks at the individual words and searches for exact matches, inflectional forms, or extensions/replacements based on the specific language's thesaurus:

```
SELECT DocumentID, DocumentSummary
FROM Production.Document
WHERE FREETEXT (DocumentSummary, 'change pedal' )
```

This returns:

DocumentID	DocumentSummary
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken components.

How It Works

In this recipe, FREETEXT was used to search the DocumentSummary column for the phrase "change pedal." Though neither the exact word "change" nor "pedal" exists in the data, a row was returned because of a match on the plural form of pedal ("pedals").

FREETEXT is, however, a less precise way of searching full-text indexes compared to CONTAINS, which is demonstrated in the next few recipes.

Using CONTAINS for Word Searching

In this recipe, I demonstrate using the CONTAINS command to perform word searches. The CONTAINS allows for more sophisticated full-text term searches than the FREETEXT predicate. The basic syntax is as follows:

```
CONTAINS
  ( { column_name | (column_list) | * } ,
'< contains_search_condition >'     [ , LANGUAGE language_term ]  )
```

The arguments are identical to FREETEXT, only CONTAINS allows for a variety of search conditions (some demonstrated later on in the "Advanced Searching" section of this chapter.

This example demonstrates a simple search of rows with a DocumentSummary searching for the words "replacing" or "pedals":

```
SELECT DocumentID, DocumentSummary
FROM Production.Document
WHERE CONTAINS (DocumentSummary, '"replacing" OR "pedals"' )
```

This returns:

DocumentID	DocumentSummary
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken components.
8	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these Adventure Works Cycles models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles parts when replacing worn or broken components.

How It Works

In this recipe, I performed a search against the DocumentSummary finding any summary that contained either the words "replacing" OR "pedals ." Unlike FREETEXT, the literal words are searched, and not the synonyms or inflectional form. Any noise words like "a" or "the" are ignored, as well as punctuation. Noise words are defined by the specified language. You can find noise word files under the SQL Server instance directory $SQL_Server_Install_Path\Microsoft SQL Server\MSSQL.1\MSSQL\FTDATA\, for filenames prefixed by "noise." For example the "noiseFRA.txt" contains French noise words.

OR was used to search for rows with either of the words, but AND could also have been used to return rows only if both words existed for the DocumentSummary value.

For a single term word, double quotes are not necessary, just the outer single quotes, for example:

```
SELECT DocumentID, DocumentSummary
FROM Production.Document
WHERE CONTAINS (DocumentSummary, 'pedals' )
```

Advanced Searching

So far this chapter has demonstrated examples of fairly straightforward word searches. However, using CONTAINS you can perform more advanced searches against words or phrases. Some examples of this include:

- Using a wildcard search to match words or phrases that match a specific text prefix.
- Search for words or phrases based on inflections of a specific word.
- Search for words or phrases based on the proximity of words near to one another.

These next three recipes will demonstrate these more advanced searches using the CONTAINS predicate.

Using CONTAINS to Search with Wildcards

In this recipe, I demonstrate how to use wildcards within a CONTAINS search. A prefix term is designated, followed by the asterisk symbol:

```
SELECT DocumentID, DocumentSummary
FROM Production.Document
WHERE CONTAINS (DocumentSummary, '"import*"' )
```

This returns:

DocumentID	DocumentSummary
7	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provided along with instructions for adjusting the tightness of the suspension fork.

How It Works

This recipe uses the asterisk symbol to represent a wildcard of one or more characters. This is similar to using LIKE, only you can benefit from the inherent performance of full-text indexing. Any match on a word that starts with "import" will be returned. In this case, one row that matches on the word "important" was returned.

When using a wildcard, the term must be embedded in double quotes; otherwise SQL Server interprets the asterisk as a literal value to be searched for. For example searching for 'import*' without the embedded quotes looks for the literal asterisk value as part of the search term.

Using CONTAINS to Search for Inflectional Matches

In this recipe, I'll demonstrate how to search for rows that match a search term based on inflectional variations. The syntax for searching for inflectional variations is as follows:

```
 FORMSOF ( { INFLECTIONAL | THESAURUS } , < simple_term > [ ,...n ] )
```

In this example, the inflectional variation of "replace" is searched:

```
SELECT DocumentID, DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, ' FORMSOF  (INFLECTIONAL, replace) ')
```

This returns:

DocumentID	DocumentSummary
3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair. Detailed instructions and illustrations are included should you need to replace the front reflector or front reflector bracket of your Adventure Works Cycles bicycle.
4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken components.
8	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these Adventure Works Cycles models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles parts when replacing worn or broken components.

How It Works

This recipe searches for any rows with the inflectional version of "replace." Although the literal value is not always found in that column, a row will also be returned that contains "replace*d*" or "replac*ing*."

THESAURUS is the other option for the FORMSOF clause, allowing you to search based on synonymous terms (which are maintained in XML files in the $SQL_Server_Install_Path\Microsoft SQL Server\MSSQL.1\MSSQL\FTDATA\ directory). For example, the French thesaurus XML file is called tsFRA.xml. These XML files are updateable, so you can customize them according to your own application requirements.

Using CONTAINS for Searching Results by Term Proximity

This recipe demonstrates how CONTAINS is used to find rows with specified words that are near one another. The syntax is as follows:

```
 { < simple_term > | < prefix_term > }
{ { NEAR | ~ }
{ < simple_term > | < prefix_term > }
```

In this example, rows are returned where the word "oil" is near to "grease":

```
SELECT DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, 'oil NEAR grease')
```

This returns:

```
DocumentSummary
Guidelines and recommendations for lubricating the required components of your
Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth
and safe ride and should be part of your standard maintenance routine. Details
instructions are provided for each bicycle component requiring regular lubrication
including the frequency at which oil or grease should be applied.
```

How It Works

This recipe looked for any text that had the word grease near the word oil.

This example searched for proximity between two words, although you can also test for proximity between multiple words, for example:

```
SELECT DocumentSummary
FROM Production.Document
WHERE CONTAINS(DocumentSummary, 'oil NEAR grease AND frequency')
```

Ranked Searching

The previous examples demonstrated full-text index searches conducted in the `WHERE` clause of a `SELECT` query. SQL Server 2005 also has ranking functions available which are referenced in the `FROM` clause of a query instead. Instead of just returning those rows that meet the search condition, the ranking functions `CONTAINSTABLE` and `FREETEXTTABLE` are used to return designated rows by relevance. The closer the match, the higher the system generated rank, as these next two recipes will demonstrate.

Returning Ranked Search Results by Meaning

In this recipe, I demonstrate `FREETEXTTABLE`, which can be used to return search results ordered by rank, based on a search string.

The syntax and functionality between `FREETEXT` and `FREETEXTTABLE` is still very similar:

```
FREETEXTTABLE (table , { column_name | (column_list) | * }
          , 'freetext_string'
     [, LANGUAGE language_term ]
     [ ,top_n_by_rank ] )
```

The two additional arguments that differentiate `FREETEXTTABLE` from `FREETEXT` are the `table` and `top_n_by_rank` arguments. The `table` argument is the name of the table containing the full-text indexed column or columns. The `top_n_by_rank` argument, when designated, takes an integer value which represents the top matches in order of rank.

In this example, rows are returned from `Production.Document` in order of closest rank to the search term "bicycle seat":

```
SELECT   f.RANK, DocumentID, DocumentSummary
FROM Production.Document d
INNER JOIN FREETEXTTABLE(Production.Document, DocumentSummary, 'bicycle seat') f
   ON d.DocumentID = f.[KEY]
ORDER BY RANK DESC
```

This returns:

RANK	DocumentID	DocumentSummary
135	8	Worn or damaged seats can be easily replaced following these simple instructions. Instructions are applicable to these Adventure Works Cycles models: Mountain 100 through Mountain 500. Use only Adventure Works Cycles parts when replacing worn or broken components.
24	3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair. Detailed instructions and illustrations are included should you need to replace the front reflector or front reflector bracket of your Adventure Works Cycles bicycle.

(Continued)

RANK	DocumentID	DocumentSummary
24	6	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth and safe ride and should be part of your standard maintenance routine. Details instructions are provided for each bicycle component requiring regular lubrication including the frequency at which oil or grease should be applied.
15	7	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provided along with instructions for adjusting the tightness of the suspension fork.
15	4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken components.

How It Works

The FREETEXTTABLE is similar to FREETEXT in that it searches full-text indexed columns by meaning, and not literal value. FREETEXTTABLE is different from FREETEXT however, in that it is referenced like a table in the FROM clause, allowing you to join data by its KEY. KEY and RANK are two columns that the FREETEXTTABLE returns in the result set. KEY is the unique/primary key defined for the full index and RANK is the measure (0 through 1000) of how good a search result the row is estimated to be.

In this recipe, the FREETEXTTABLE result set searched the DocumentSummary column for “bicycle seat,” and joined by its KEY value to the Production.Document table’s DocumentID column:

```
INNER JOIN FREETEXTTABLE(Production.Document,
DocumentSummary,
'bicycle seat') f
   ON d.DocumentID = f.[KEY]
```

RANK was returned sorted by descending order, based on the strength of the match:

```
ORDER BY RANK DESC
```

Returning Ranked Search Results by Weighted Value

In this recipe, I demonstrate returning search results based on a weighted pattern match value using the CONTAINSTABLE command. CONTAINSTABLE is equivalent to FREETEXTTABLE in that it acts as a table and can be referenced in the FROM clause. CONTAINSTABLE also has the same search capabilities and variations as CONTAINS.

Both CONTAINS and CONTAINSTABLE can be used to designate a row match’s “weight,” giving one term more importance than another, thus impacting result rank. This is achieved by using ISABOUT in the command, which assigns a weighted value to the search term.

The basic syntax for this is as follows:

```
   ISABOUT { <search term> }   [ WEIGHT ( weight_value ) ]
```

This example demonstrates querying Production.Document by rank, giving the term “bicycle” a higher weighting than the term “seat”:

```
SELECT f.RANK, d.DocumentID, d.DocumentSummary
FROM Production.Document d
INNER JOIN CONTAINSTABLE(Production.Document, DocumentSummary,
'ISABOUT ( bicycle weight (.9), seat weight (.1))') f
   ON d.DocumentID = f.[KEY]
ORDER BY RANK DESC
```

This returns:

RANK	DocumentID	DocumentSummary
23	3	Reflectors are vital safety components of your bicycle. Always ensure your front and back reflectors are clean and in good repair. Detailed instructions and illustrations are included should you need to replace the front reflector or front reflector bracket of your Adventure Works Cycles bicycle.
23	6	Guidelines and recommendations for lubricating the required components of your Adventure Works Cycles bicycle. Component lubrication is vital to ensuring a smooth and safe ride and should be part of your standard maintenance routine. Details instructions are provided for each bicycle component requiring regular lubrication including the frequency at which oil or grease should be applied.
11	7	It is important that you maintain your bicycle and keep it in good repair. Detailed repair and service guidelines are provided along with instructions for adjusting the tightness of the suspension fork.
11	4	Detailed instructions for replacing pedals with Adventure Works Cycles replacement pedals. Instructions are applicable to all Adventure Works Cycles bicycle models and replacement pedals. Use only Adventure Works Cycles parts when replacing worn or broken components.

How It Works

The CONTAINSTABLE is a result set, joining to Production.Document by KEY and DocumentID. RANK was returned in the SELECT clause, and sorted in the ORDER BY clause. CONTAINSTABLE can perform the same kinds of searches as CONTAINS, including wildcard, proximity, inflectional, and thesaurus searches.

In this example, a weighted term search was performed, meaning that words are assigned values that impact their weight within the result ranking.

In this recipe, two words were searched, "bicycle" and "seat," with "bicycle" getting a higher rank than "weight":

```
'ISABOUT ( bicycle weight (.9), seat weight (.1))'
```

The weight value can be a number from 0.0 through 1.0 and impacts how each row's matching will be ranked within CONTAINSTABLE. ISABOUT is put within the single quotes and the column definition is within parentheses. Each term was followed by the word "weight" and the value 0.0 to 1.0 value in parentheses.