

The full-text search built into SQL Server 2005 is a hugely powerful feature that can improve the performance of your databases. With an index that's remarkably like the one you'll find in the back of a reference book, your users can find the content they seek swiftly, by concept as well as by spelling. A search for "color" can find "colour," if you like, and the keyword "chocolate" can return results that include "cocoa."

Search me: using SQL Server full-text search

INSIDE

- **02** **FTS**
advantages
- **09** **Create a full-**
text catalog
- **11** **Tables using**
T-SQL
- **17** **Indexes using**
the Wizard
- **26** **XML in FTS**
- **33** **Testing FTS**

Search me: using SQL Server full-text search

By Hilary Cotter

Have you ever used the like operator to search columns for a string? While the like operator is very accurate, it simply is not scalable.

Should you use like to conduct searches on your Web sites, you will find that these searches cause frequent timeouts and can degrade overall database performance.

Your customers may become impatient with the length of time their searches take. They might leave for a competitor's Web site, where their search solution performs much faster - probably

using a full-text search engine, such as SQL Full-Text Search (SQL FTS), which is the focus of this article.

Full-text indexes are conceptually similar to the indexes you find at the back of a book; they help you find pages where a word or a phrase occurs. SQL FTS allows you to find the database rows which contain the word or phrase for which you are looking.

SQL FTS is an optional component which was first available in SQL 7 Beta 3. Technically speaking, the component allows you to build

“Full-text indexes are conceptually similar to the indexes you find at the back of a book; they help you find pages where a word or a phrase occurs.”



SQL FTS allows you to find the database rows which contain the word or phrase you are looking for.

indexes on columns of the char, varchar, text, image (SQL 2000 and above), varchar(max) (SQL 2005), varbinary(max) (SQL 2005), and XML (SQL 2005) data type columns. You can query them in SQL Server using the Contains, ContainsTable, FreeText, or FreeTextTable predicates.

Using full-text search gives your applications several advantages. Among them:

- ✦ The queries are much faster than if you were to use the like predicate. For larger SQL tables, full-text search may be many orders of magnitude faster than using a like predicate.

“You can store documents in SQL Server in their binary form and search for that data later.”

- ✦ You can incorporate language features in your queries to incorporate searches on all generations of a word. By “generations of a word,” Microsoft means both thesaurus features and stemming for declensions and conjugations of a word. Declension includes singular and plural forms as well as masculine and feminine forms. For example, a search for apple also matches with apples. Conjugation means all tenses of a word; a search on run matches ran and running.
- ✦ Multiple language support. For example, even in an English-language version of SQL Server, you could search on Chinese characters.
- ✦ You can incorporate wild-

carding and weighting of phrases.

- ✦ You can store documents in SQL Server in their binary form and search for that data later. For example, if you store Word documents in your tables, you can search for words and phrases in those documents, as if they were pure text documents stored in the columns.

Microsoft was an innovator in delivering the first desktop search solution; it was Bill Gates personal initiative, first announced to the world in that now-famous 1990 Comdex keynote address, Information at your Fingertips. As Microsoft has been a player in the search space for so long, they learned important lessons.

One of the most important lessons learned is that people expect their searches to incor-

porate features of their spoken and written language. They expect their searches to return results to the word, and also to plural or other verb forms of that word. So they expect a search on book to return results not only containing book, but also books. Users also expect searches on run to return results to ran, runs, running, etc.

Another lesson that Microsoft learned was the considerable variation in spelling within the same language. Searches have to be conceptual searches as well as strict searches; a search for center might miss

content where the author spelled the word as centre, and a search on pants might miss content where the term trousers was used.

Microsoft addressed these language problems by developing rich language-aware features in their search products, including SQL FTS.

Before I explain how to use SQL full-text search, let me give you a short introduction to its architecture.

SEARCH ARCHITECTURE

It's likely that you've encountered some parts of Microsoft's search features in other products. The architecture of SQL FTS is based on Site Server Search, which in turn was based on Index Server (now known as Indexing Services). The search engine used in SQL FTS is similar to the engine used for Office XP Search (MOSearch), Site Server Search, SharePoint Portal

Search, Exchange Content Indexing, and Windows Desktop Search. SharePoint Team Services search uses SQL Server Full-Text Search.

Site Server Search starts with a URL to a Web page. It crawls all hyperlinks in that Web page, returning them to the indexing component, which then creates an index. SQL FTS still uses this crawl concept, crawling your tables to extract each row one by one, then indexing them. The crawl process is called a population, as your index is being populated with data.

SQL FTS has five primary components. We'll explore each of them briefly.

- 1 Gatherer
- 2 Indexer
- 3 Filter Manager
- 4 Filter Daemon
- 5 Full-Text Catalog



Microsoft has addressed the problem of discrepancies in language by developing rich language aware features in their search products, including SQL FTS.

A System of Excellence

The Kenton County School District



An IT infrastructure based on Dell platforms keeps the Kenton County School District at the head of the class

The Kenton County School District is a big district with an even bigger reputation for excellence. Located in northern Kentucky, it serves over 13,000 students throughout 21 schools and has reaped many honors, including being selected as a spotlight school district by the U.S. Department of Education in 1999 and 2000, and receiving the national “What Parents Want” award for 12 years in a row.

Maintaining such a high level of excellence in academics requires excellent IT. In order to support classroom activities, the district’s administrative functions, transportation, food service, and finance must run reliably and efficiently. Also, critical information such as student records, class schedules, and other data must be collected, maintained, and made readily available to teachers and other staff who require it. “We need an IT infrastructure that can support all of our departments,” says Vicki Fields, district technology coordinator for the Kenton County School District. “Plus, we want our teachers to have instant access to student information so they can spend their time teaching instead of chasing down student records.”

“Without question, Dell keeps our applications up and running better than any other vendor with whom I have worked in the past.”

— **Vicki Fields**, District Technology Coordinator
Kenton County School District

A System of Excellence

The Kenton County School District



Kenton County looks for a new platform

A few years ago, the district wanted to improve teacher and staff access to student information and other types of data. “We have information distributed across a variety of delivery systems,” says Fields. “We wanted to provide comprehensive access to that information to better facilitate teaching, administrative functions, and decision making throughout the district.”

Also, Fields wanted to standardize on a single hardware vendor. “We have many different applications throughout the district, and we wanted to find one vendor that could support them all,” says Fields. “In addition, we wanted a vendor with excellent support and services that would be with us over the long haul.”

Kenton County decides on Dell

After evaluating several vendors, Fields was impressed with the flexibility of Dell offerings. “Other off-the-shelf types of technologies didn’t have the flexibility that we needed,” says Fields. “When I approached Dell, their willingness to comply with our requirements was a key selling point. Dell told us we could set up our system however we wanted, and that was just what we needed to hear.”

To view the entire story, go to www.dell.com/sql

GATHERER

The Gatherer component connects to SQL Server, extracts the rows from the tables which you are full-text indexing, and sends them to the Indexer. In SQL 2005, the Gatherer runs in the SQL Server process. In SQL 2000, it is part of MSSearch.

INDEXER

As its name implies, the indexer performs the index functions on the text stream sent from the Filter Daemon process (MS-FTED: Microsoft SQL Full-Text Engine Daemon) and stores it in the catalog. The Indexer stores temporary indexes in memory until it can flush them to disk.

These indexes are called word lists. When a specified number of word lists are in RAM (you can control this number with `sp_fulltext_service 'resource_usage'`) they are consolidated in a shadow index. When you perform a reorganize, the shadow indexes are merged into a single index called a master index. The master index is stored in the catalog.

FILTER MANAGER

The Gathering Manager is a supervisor: it manages the indexing and crawl processes, and keeps an error log. If the

Indexer cannot keep up with the crawl process, the Filter Manager throttles the crawling process until the indexer can catch up. The Filter Manager then increases the crawl rate until a happy medium is met.

FILTER DAEMON

The Filter Daemon process extracts textual content from the data stream emitted from the Gatherer, and breaks this textual content into words or tokens. A token is a string of characters which may or may not have linguistic significance, such as and, XXX, MSFT, qwerty, and rytrrrt.

The Filter Daemon (MSFTEFD) does this by implementing a COM component called an iFilter. The iFilter discerns the content's file format, and emits only the text to the Indexer; it breaks the text apart at word boundaries, using another COM component

“The Gathering Manager is a supervisor: it manages the indexing and crawl processes, and keeps an error log. If the Indexer cannot keep up with the crawl process, the Filter Manager throttles the crawling process until the indexer can catch up.”

called a word breaker. The word breaker used is determined by:

- ✦ The language type you specified
- ✦ The default full-text language setting for your copy of SQL Server (obtainable by issuing a `sp_configure` 'default full-text language')
- ✦ The embedded language type in the document you stored in SQL Server, assuming that the iFilter

“By default, SQL FTS is installed in all editions of SQL 2005, except SQL 2005 Express and SQL Server Mobile editions.”

respects these language types; HTML, Word, and XML iFilters are language-aware. You can check for embedded language tags by using the `FiltDump` utility, which ships with the Platform SDK.

You can obtain more information on Word Breakers and iFilters by referring to “Implementing a Word Breaker” <http://msdn2.microsoft.com/en-us/library/ms693186.aspx>. I'll go into more detail about Word Breakers and Languages in the section on setting up Full-Text Search section.

FULL-TEXT CATALOG

The full-text catalog contains the indexes for one or more full-text indexed tables. If you full-text index a table, you can store the full-text index in only one catalog; it cannot span catalogs.

INSTALLING THE FULL-TEXT SERVICE

By default, SQL FTS is installed in all editions of SQL 2005, except SQL 2005 Express and SQL Server Mobile editions. If it is not installed, you have to run setup again and select it in the Database Engine component. You also need to enable SQL FTS using SQL Server 2005 Surface Area Configuration. Click on Surface Area Configuration for Services and Connections, then choose Full-Text Search. Change Startup Type to Automatic.

If you are upgrading SQL 2000, the full-text components are upgraded. The catalog is repopulated as part of the upgrade process.

Although it isn't included by default, you can install SQL full-text search on SQL 2005 Express. To do so, download the Advanced Services for SQL 2005 Express by clicking on

this link <http://go.microsoft.com/fwlink/?LinkId=65109>.

CREATING FULL-TEXT INDEXES

Creating full-text indexes on your tables is a three step process:

- 1 Enabling your database for full-text indexing
- 2 Creating a full-text catalog
- 3 Creating full-text indexes on your tables

There are two methods of performing each step: through the Wizard, and through TSQL. We'll go into each method in turn.



If your tables are large, they should have a dedicated catalog for optimal performance. This size depends on your machine, and the amount of content in each row.

ENABLING DATABASES FOR FULL-TEXT INDEXING

Before you create full-text indexes on your tables, you must enable the database for full-text indexing. To do this, you can issue the below SQL statement:

```
sp_fulltext_database 'enable'
```

You can also achieve the same thing by using SQL Server Management Studio. Expand the Databases folder, right click on your database, then select Properties. In the Files tab, ensure that the "Use full-text indexing" check box is checked.

CREATING A FULL-TEXT CATALOG

The next step in the process is to create a full-text catalog; the indexes have to have somewhere to go! To create the catalog, you can use a command like the one below:

```
Create FullText Catalog  
VictoriaSecret
```

Where my catalog name is **VictoriaSecret**. (Naturally you can call your catalog any name you want, but the Victoria Secret catalog is a very popular one!)

The creation command has a few options. For example, if you want this catalog to be the default catalog for all full-text indexes in your database, you would issue:

```
Create FullText Catalog  
CatalogName As Default
```

If your tables are large, they should have a dedicated catalog for optimal performance. How large is large? This depends on your machine, and the amount of content in each row. In a multi-terabyte search solution, we implemented, we found a sweet spot exists for querying and indexing performance around 50 million rows on a Quad (DL 585). We currently have 8 full-text catalogs to de-

liver optimal performance. Your mileage may vary.

By its default behavior, the full text search is accent sensitive; it recognizes *café* and *cafe* as two different words. To instruct the index function to ignore accents, tell it so:

```
Create FullText Catalog
CatalogName with Accent_
Sensitivity=OFF
```

If you detach or backup your database, you have the option of storing the catalog in your database. Query `sys.databasefiles`, and you will notice a row with a `type_desc` value of FULLTEXT. This is the database file in which the catalog will be stored if you backup the database or detach it. The advantage of this is that you can backup the catalogs at a different frequency from your databases if you backup your database by file group, or you can do file group backups which

will not include the full-text catalog and consequently be much smaller.

Note that if you are using change tracking your catalog may be considerably out of date from your database, so this option only works if you are doing incremental or full populations. If you are using change tracking you should do full database backups, which will include all of your file groups along with your full-text catalogs.

To enable the separate file store, issue this command:

```
Create FullText Catalog
CatalogName with on
FileGroup 'FullText'
```

Where FullText is the name of the filegroup to dedicate for the full-text backup.

By default, your full-text catalogs are created in the full-text path for SQL Server, generally `C:\Program Files\Microsoft SQL Server\MSSQL.X\MSSQL\FTDa-`

`ta`. You can change the default location by modifying the registry entry `FullTextDefaultPath` in the key `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.X\MSSQLServer`.

To place your full-text catalog in a different location, select the path using the following command:

```
Create FullText Catalog
CatalogName in path 'c:\
temp'
```



By its default behavior, the full text search is accent sensitive; it recognizes *café* and *cafe* as two different words. But you can instruct the index function to ignore accents.

For performance reasons, you should place your catalog on the fastest disk subsystem possible – ideally on a dedicated RAID 10 array with a dedicated controller.

CREATING FULL-TEXT INDEXES ON TABLES USING T-SQL

Creating full-text indexes on your tables in SQL Server 2005 is symmetrical with creating table or indexed view statements; you use the same `Create`, `Al-`

`ter`, and `Drop` statements. For backwards compatibility, the `sp_fulltext_table` and `sp_fulltext_column` from SQL 2000 and SQL 7 are still supported.

The syntax to create an index looks like this:

```
Create FullText Index on
TableName (ColumnName1,
ColumnName2) Key Index
TableNamePrimaryKey
```

Where `TableName` is the table name to index, and `ColumnName1` and `ColumnName2` are the `char`, `varchar`, `XML`, or `text` datatype columns to index. `TableNamePrimaryKey` identifies the name of the primary key or the name of a unique index on your table. The index you select must be a single column index or a unique index. (We will cover how to index `Image` and `VarBinary(max)` datatype columns shortly.) The example above only works if you already have a full-text

catalog defined by default for this database. Otherwise, you get the following error message:

```
Msg 7623, Level 16, State 1,
Line 1
```

A default full-text catalog does not exist in database 'DatabaseName' or user does not have permission to perform this action.

Without a default full-text catalog defined, your syntax has to look like this:

```
Create FullText Index on
TableName (ColumnName1,
ColumnName2) Key Index
TableNamePrimaryKey on
CatalogName
```

In all the examples above, all columns are indexed using the word breakers for the language you set for SQL Server, or whatever you specified as the default full-text language for SQL Server. That is, if you had performed a default installation

“In these examples, all columns are indexed using the word breakers for the language you set for SQL Server, or whatever you specified as the default full-text language for SQL Server.”

of SQL Server using US English, the US English word breakers would be used.

You can, of course, change the value for the default full-text language setting on this server. To set the default full-text language setting, issue the following command:

```
Sp_configure 'show advanced
options',1
reconfigure
sp_configure 'default full-text
language',1033
GO
```



To index columns in your tables using different language word breakers, you can specify the language in the full-text index creation statement.

In this example, we set the default language for indexing full-text indexed columns to US English. The final value is the language code identifier (LCID), 1033 in the example—which may not be a number that springs immediately to mind.

To correlate the LCID with the language you are using, query syslanguages:

```
SELECT alias, LCID FROM sys.
syslanguages
```

Or, you can consult Table 1 for a list of the most common languages supported. I obtained this list from querying:

```
select * from sys.fulltext_
languages
```

If you need additional language support - there are six I didn't include here - consult "Licensed Third-Party Word Breakers" <http://support.microsoft.com/kb/908441/en-us>
To index columns in your tables

**TABLE 1:
SUPPORTED LANGUAGES IN SQL 2005**

LCID	LANGUAGE
2052	Simplified Chinese
1028	Traditional Chinese
1031	German
2057	British English
1033	English
3082	Spanish
1036	French
1040	Italian
1041	Japanese
1042	Korean
0	Neutral
1043	Dutch
1053	Swedish
1054	Thai
3076	Chinese (Hong Kong SAR, PRC)
5124	Chinese (Macau SAR)
4100	Chinese (Singapore)

using different language word breakers, you can specify the language in the full-text index creation statement. Here is an example:

```
CREATE FULLTEXT
INDEX on TableName
(USEngishColumn
LANGUAGE 1033,
JapaneseColumn
LANGUAGE 1041)
```

The final part of the full-text index creation statement we examine is the population type clause. The population is the process which extracts the textual data from your table

“Change tracking is a background process which keeps your full-text index updated with inserts, updates, or deletes that occur on the table during the time it takes for the full-text indexing.”

and creates the full-text index. By default, whenever you create a full-text index on a table using T-SQL, a population is run and *change tracking* is enabled. Change tracking is a background process which keeps your full-text index updated with inserts, updates, or deletes that occur on the table during the time it takes for the full-text indexing.

The full-text population options are:

- ✦ **AUTO:** change tracking with update full-text index in background. This is the default.
- ✦ **MANUAL:** change tracking is turned on, but you update the full-text index according to your schedule.
- ✦ **OFF:** change tracking is not enabled, but a full population is started when the full-text index is created.

- ✦ **OFF, NO POPULATION:** change tracking is not enabled, and a full population is not performed as soon as the full-text index is created.

That’s a little dry. Let’s look at an example of each type:

AUTO

```
CREATE FULLTEXT INDEX on
MyTable (charcol) KEY INDEX
PrimaryKey ON Catalog_
Name WITH CHANGE_
TRACKING AUTO
```

or

```
CREATE FULLTEXT INDEX on
MyTable (charcol) KEY INDEX
PrimaryKey ON Catalog_
Name
```

MANUAL

Sometimes change tracking causes locking on the tables you are full-text indexing. In this case, you may want to schedule manual population:

```
CREATE FULLTEXT INDEX on
MyTable (charcol) KEY INDEX
PrimaryKey ON Catalog_
Name WITH CHANGE_
TRACKING MANUAL
```

After the full-text index is created, SQL Server will run a population, but the catalog is not updated with changes to your table after the population is complete. If you choose the manual option, you have to update the index to keep it up to date. Use the following command for this:

```
ALTER FULLTEXT INDEX
ON MyTable START UPDATE
POPULATION
```

OFF

In some cases, you use tables with static data which do not change, such as a list of U.S. states. Or, if a large portion of the table is updated in a batch, you may wish to disable change tracking completely. Nonetheless, you still want a full popu-

lation to be run immediately. Here is the syntax to do that.

```
CREATE FULLTEXT INDEX on
MyTable (charcol) KEY INDEX
PrimaryKey ON Catalog_
Name WITH CHANGE_
TRACKING OFF
```

OFF, NO POPULATION

After you create your full-text index, a full population is run. This may cause locking on the base table and you may want to disable the initial full population and have it run during a quiet time. Here is the syntax to do this:

```
CREATE FULLTEXT INDEX
on MyTable (charcol)
KEY INDEX PrimaryKey
ON Catalog_Name WITH
CHANGE_TRACKING OFF, NO
POPULATION
```

When you are ready want to run the full population, issue the statement:

```
ALTER FULLTEXT INDEX
ON MyTable START FULL
POPULATION
```

ALTER FULLTEXT INDEX

We've already seen the Alter FullText index statement. However, some of its additional features can be useful. You can use the Alter FullText index command to:

- ✦ [Add/drop columns](#)
- ✦ [Enable/disable the index on a full-text indexed table](#)
- ✦ [Control your population](#)

ADDING/DROPPING COLUMNS

You can use the [Alter FullText Index](#) statement to add additional columns to full-text index, or to drop columns you no longer wish to have full-text indexed.

Here is an example of how to add a column:

```
ALTER FULLTEXT INDEX
MyTable ADD (NewCol)
```

And to drop a column:

```
ALTER FULLTEXT INDEX  
MyTable Drop (Col)
```

ENABLE/DISABLE

You can also use the `Alter Full-Text Index` statement to enable or disable a table's full-text index.

When you issue a disable command, any population in progress pauses. If you re-enable the full-text index, change tracking resumes. However, full and incremental populations do not continue; you have to re-start them manually. Here's an example:

```
ALTER FULLTEXT INDEX ON  
MyTable ENABLE
```

CONTROLLING POPULATION

Population control sounds like a theme from a science fiction book like *Stand on Zanzibar*. But in this context, we're talking about the SQL Server process to update full-text indexes.

Use `Alter FullText Index` to control your populations. For example, to stop a population:

```
ALTER FULLTEXT INDEX on  
MyTable Stop Full Population
```

To start a full population, use this command.

```
ALTER FULLTEXT INDEX on  
MyTable Start Full Population
```

Or to run an incremental population:

```
ALTER FULLTEXT INDEX on  
MyTable Start
```

INCREMENTAL POPULATION

An *incremental population* is similar to a full population. In an incremental population, every row is extracted from the table and compared with what is in the catalog. New entries are added, deleted rows are removed from the catalog, and updated rows are updated in the catalog.

An incremental population

requires a timestamp column on the table you are full-text indexing; otherwise, a full population is run. If at all possible, use change tracking, because an incremental population takes almost as long as a full population—and in some cases, it may take longer.

DROP FULLTEXT INDEX

The final command is the `Drop FullText Index` statement. Use this statement to drop your full-text indexes:

```
Drop FullText Index on  
TableName
```



If at all possible, use change tracking, because an incremental population takes almost as long as a full population—and in some cases, it may take longer.

FULL-TEXT INDEXING INDEXED VIEWS

In SQL Server 2005, it is possible to create a full-text index for views. This is a very valuable feature, particularly if you work with large datasets. When a table gets large, you may need to partition it into ranges when you need to trim your queries. For example, consider this query:

```
Select * from Resumes  
where Location='NE' and  
Contains(Skills,'SQL')
```

In this query, the full-text engine returns all results from the full-text catalog to SQL server and then trims the results to records where the location is NE.

Full-text queries perform best for small results sets, typically under 2,000. For large results sets, the query may take a long time. The length of the query depends on the complexity of the query and the number of

matches. Some of my more complex queries which have returned a large amount of rows (40,000) have taken up to 45 minutes. In general, if you limit your results to 200 rows, you will get sub-second response for most queries. If you could create a index view which only returns locations of NE, and full-text index this view, performance would be much better:

```
CREATE Database Resumes
```

```
GO
```

```
Use Resumes
```

```
GO
```

```
CREATE TABLE Resumes (PK int identity constraint ResumesPK  
primary key, Skills VARCHAR(100), Location char(2))
```

```
GO
```

```
INSERT Resumes (Skills, Location) VALUES ('SQL','NE')
```

```
INSERT Resumes (Skills, Location) VALUES ('SQL','NW')
```

```
INSERT Resumes (Skills, Location) VALUES ('NT','NE')
```

```
INSERT Resumes (Skills, Location) VALUES ('NT','NW')
```

```
GO
```

```
CREATE fulltext catalog cat1
```

```
GO
```

```
CREATE VIEW myview WITH SCHEMABINDING AS
```

```
SELECT PK, Skills, Location FROM dbo.Resumes
```

```
WHERE
Location='NE'
GO
-- create the index on the view to be used as fulltext key index
CREATE UNIQUE CLUSTERED INDEX idx ON myview(PK)
GO
CREATE fulltext INDEX ON myview(Skills) KEY INDEX idx ON cat1
GO
WHILE fulltextcatalogproperty('cat1','populatestatus') <> 0
BEGIN
    WAITFOR DELAY '00:00:01'
END
GO
-- note only results are returned where skill=sql and location=ne
SELECT * FROM myview WHERE CONTAINS(Skills,'SQL')
GO
```



The SQL FTS Wizard is simple to use and quite intuitive—as long as you have a vague idea of what you are doing. For a higher degree of control, you should use the T-SQL commands covered above.

CREATING FULL-TEXT INDEXES USING THE WIZARD

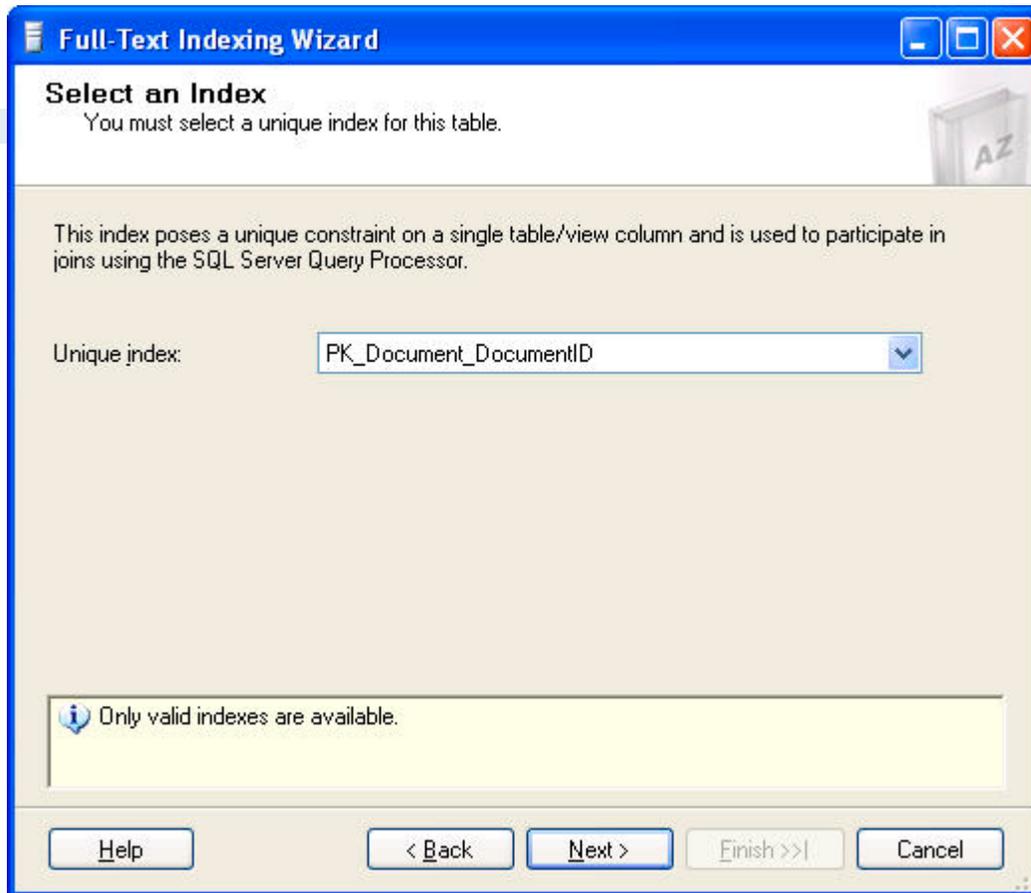
Microsoft spends millions consulting with UI design engineers to make their products as simple and intuitive to use as possible. SQL FTS is no exception. Microsoft selected defaults which provide optimal performance for most of their customers. Chances are very good that the SQL FTS Wizard will work well for you!

The Wizard is simple to use and quite intuitive—as long as you have a vague idea of what you are doing. However, not all of the features are exposed in the Wizards. For a higher degree of control, you should use the TSQL commands covered above.

The first step, before you index tables using the Full-text Index Wizard in SQL Server 2005 Management Studio, is to make sure your database has the feature enabled. To do this,

1

FIGURE 1: SELECTING AN INDEX

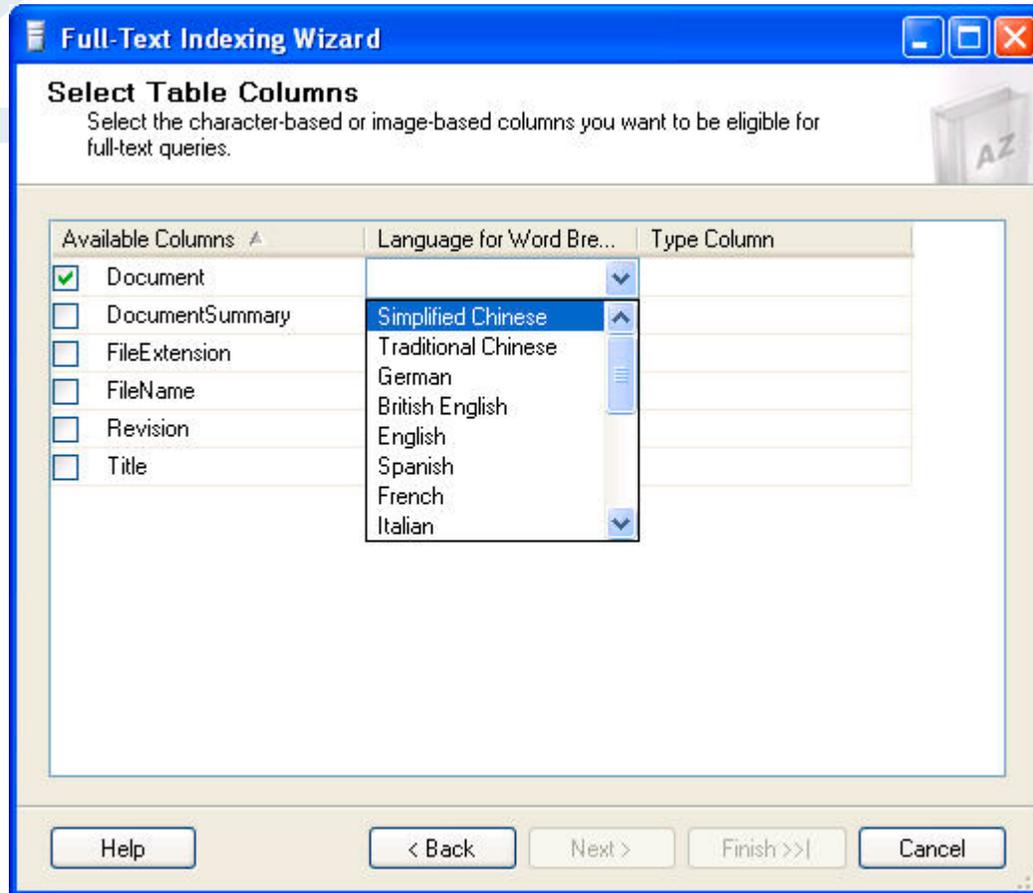


connect to your SQL Server in SQL Server Management Studio, expand your server node, expand the databases node, and right click on your the database; select properties; select the Files tab; and ensure the “use full-text indexing” check box is checked.

After this has been done, right click on the table to full-text index, and select Full-Text index. Then select Define Full-Text Index. After you see the full-text index splash screen (click on Next), you are prompted to select a unique index to use as the key. The Wizard will auto detect candidate keys. This dialog is illustrated in Figure 1.

Select the appropriate index. Ideally, choose the narrowest one which will be as static as possible. Click on Next. In the Select Table Columns dialog box (shown in Figure 2), select the columns to full

FIGURE 2: SELECTING TABLE COLUMNS



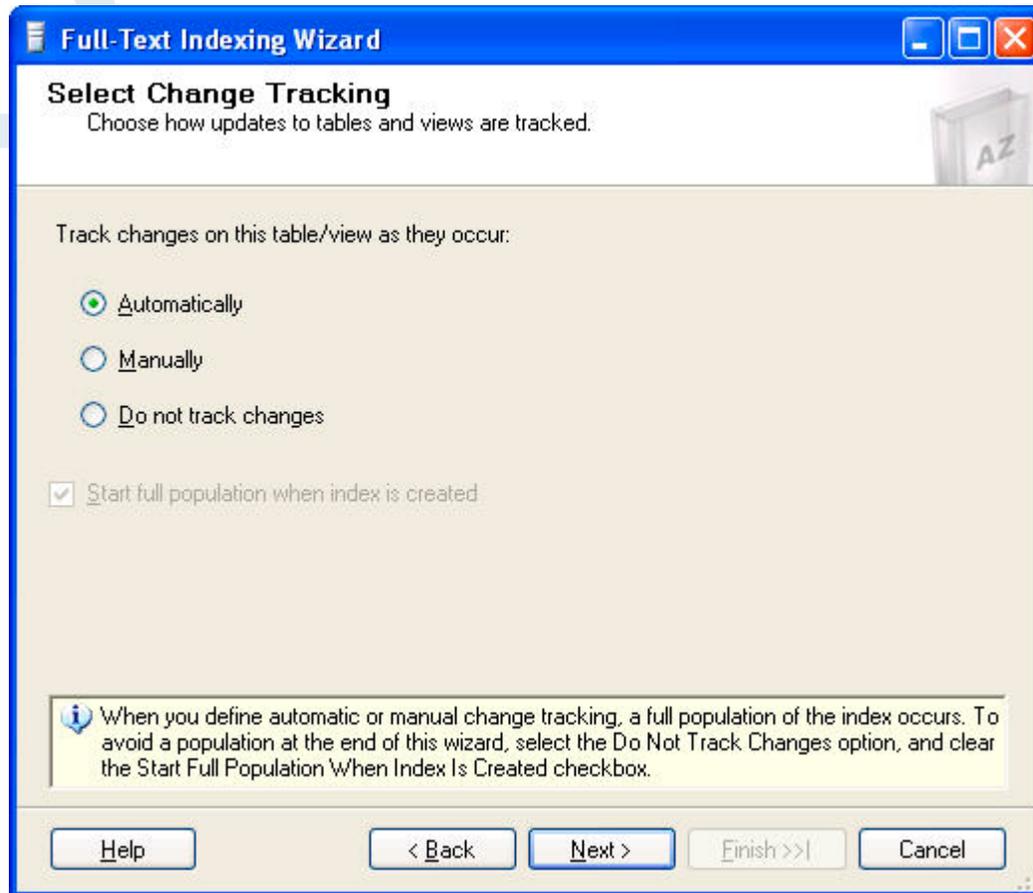
text index. Notice that you can select the word breaker that SQL Server should use to index this column. You do not have to make a language selection if you want SQL Server to use its default full-text language setting.

In the table shown in this example, the document column is varbinary. Thus, the wizard does not let you click Next until you select a Type column. This is because the column type is an **Image** or **VarBinary** data type column, and contains a **blob**. (We will cover this in more detail in the next section, Indexing Blogs.)

Click on Next to select Population options, as illustrated in Figure 3.

Normally, the best option to choose for change tracking is "Automatically." However, change tracking can cause locking, so you might want to select Manual; it does not provide the

FIGURE 3: THE SELECT CHANGE TRACKING DIALOG BOX.



real time full-text indexing that change tracking does.

Changes are still tracked, but the index is only updated when you specify it. Use the Apply Tracked Changes menu option, as shown in Figure 4.

If you select “Do not track changes,” a full population is performed as soon as you complete this dialog. However, a full population can cause locking, so you might want to delay the full population to a quiet time.

FIGURE 4: SELECTING APPLY TRACKED CHANGES

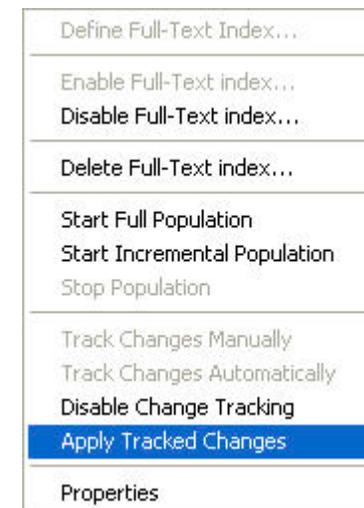
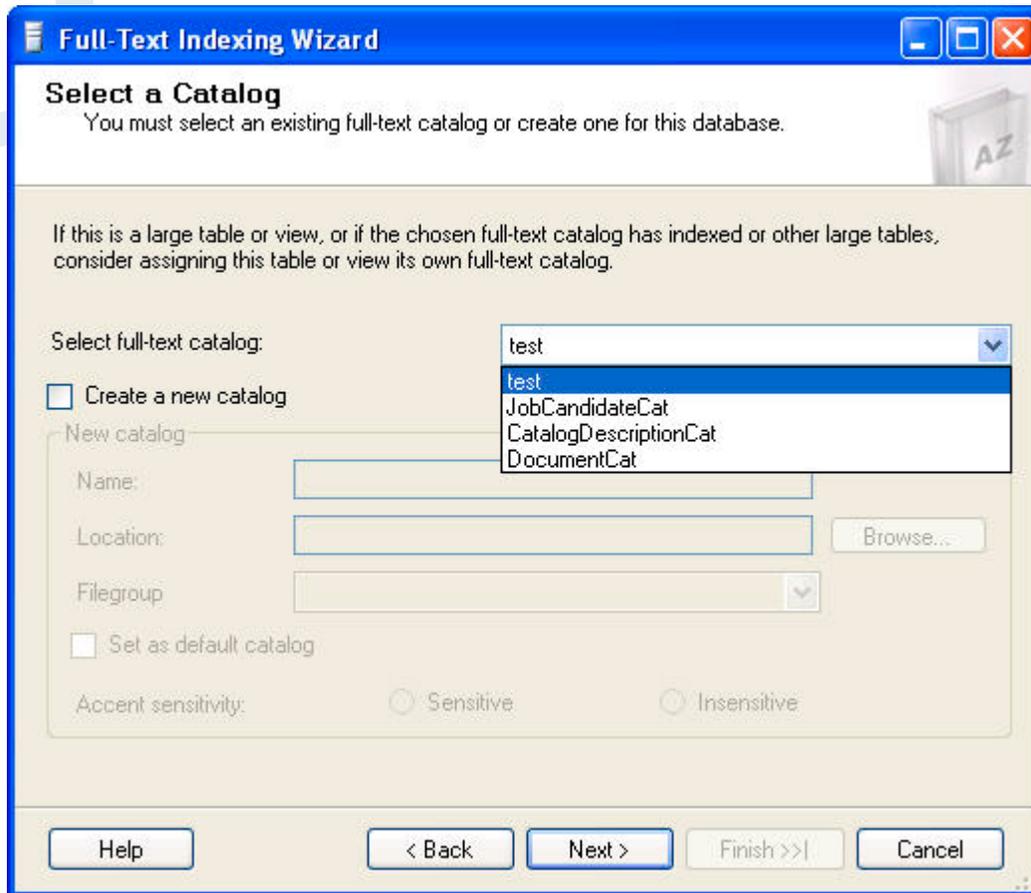


FIGURE 5: SELECT A CATALOG DIALOG BOX



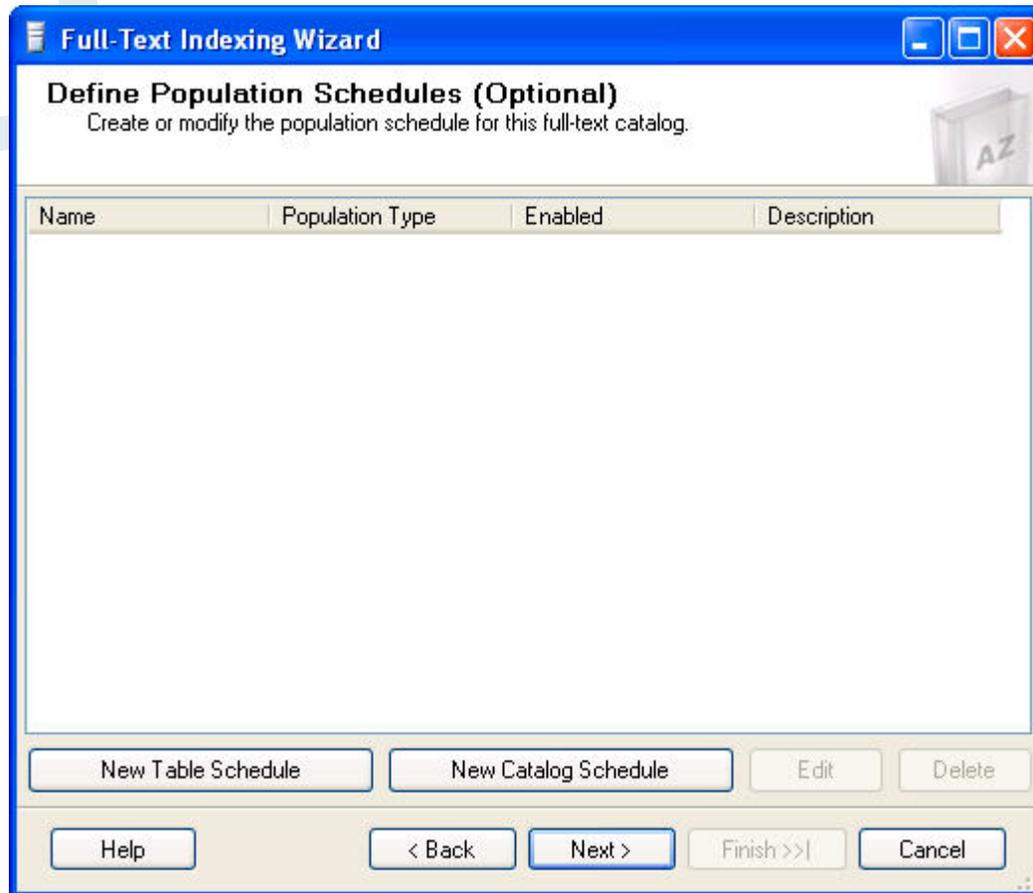
If so, uncheck the “start full population when index is created” check box.

Returning to the Select Change Tracking dialog box, click on Next to advance to the Select a Catalog dialog box, as illustrated in Figure 5.

In this dialog, click in the drop down list box to choose an existing full-text catalog in which to place your full-text index. Alternatively, you can create a new full-text index by clicking the Create a new catalog dialog box. This option allows you to:

- ✦ Name the catalog,
- ✦ Select where you want to place the catalog,
- ✦ Choose the filegroup in which you wish to place the catalog,
- ✦ Set the newly created catalog as the default catalog, and
- ✦ Set the accent sensitivity.

FIGURE 6: THE DEFINE POPULATION SCHEDULES DIALOG



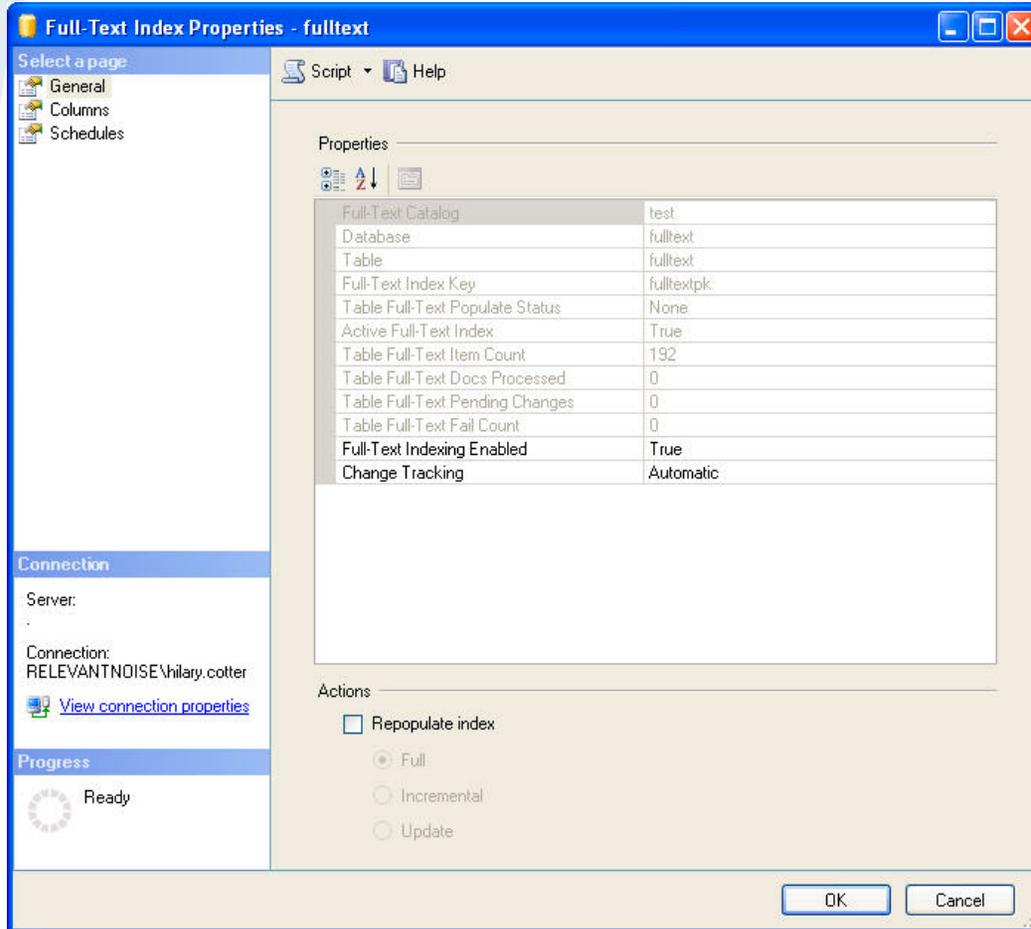
When you have selected or created the catalog to use, click Next to advance to the next dialog box.

This dialog, shown in Figure 6, allows you to define the frequency with which your full-text indexed table should be populated (using the New Table Schedule button), and how frequently you wish all tables in your catalog to be populated (using the New Catalog Schedule button).

With change tracking, you do not need to define a population schedule. You only need to do so if you use the “do not track changes” option in the Select Change Tracking dialog illustrated in Figure 3.

Click on Next for a summary page dialog, and then Finish to build your full-text index. Once you click Finish, your full-text index is created. Depending on the population option you selected, your population may

FIGURE 7: FULL-TEXT INDEX PROPERTIES



be started. You can start querying your full-text indexes momentarily, but they may not be completely populated yet.

To check the population status, right click on the table in SQL Server Management Studio and select Full-Text index and then Properties. You will see the dialog illustrated in Figure 7.

To find out how far your population process has progressed, examine the Table Full-Text Populate Status column and the Table Full-Text Docs Processes. In this example, the Table Full-Text Populate Status reads "none," indicating that the population is not active. Possible values are idle, processing, background, and none.

The Table Full-Text Docs Processed shows the number of rows indexed. You can compare this value with the number of rows in your table to determine your population status.

INDEXING BLOBS WITH IFILTERS

As previously mentioned, you can store documents in their native format in `varbinary(max)` (in SQL 2005) and `image` (in SQL 2000 and SQL 2005) data type columns and have them indexed. For example, you can store Word, Excel, or PDF documents in these columns, and SQL Server Full-Text will index them.

You must establish another column in the table to contain the extension the document would have if you were storing it in the file system; that is, a Word document could be stored in the blob, and the secondary column could store

“doc”. If you do so, you can tell the full-text index to reference the file-type column to determine the document type and to load the correct iFilter. If you don't, the Indexer (actually the Filter Daemon) will try to index the contents of the row using the default (text) iFilter.

Remember: iFilters are COM components which the Indexer uses to extract textual data from binary file formats, such as like TIFFs, PDFs, and Word documents. To get a list of the iFilters which ship with SQL Server issue the following query:

```
select * from sys.fulltext_
document_types
```

If the document type to index is not there, you need to install it on your SQL Server computer. For a list of available iFilters, consult <http://www.index-serverfaq.com/iFilters.htm> Once you acquire the new iFilters, you need to issue these two commands so that SQL Server can use them:

```
sp_fulltext_service 'load_os_
resources'
```

```
sp_fulltext_service 'verify_
signature', 1
```

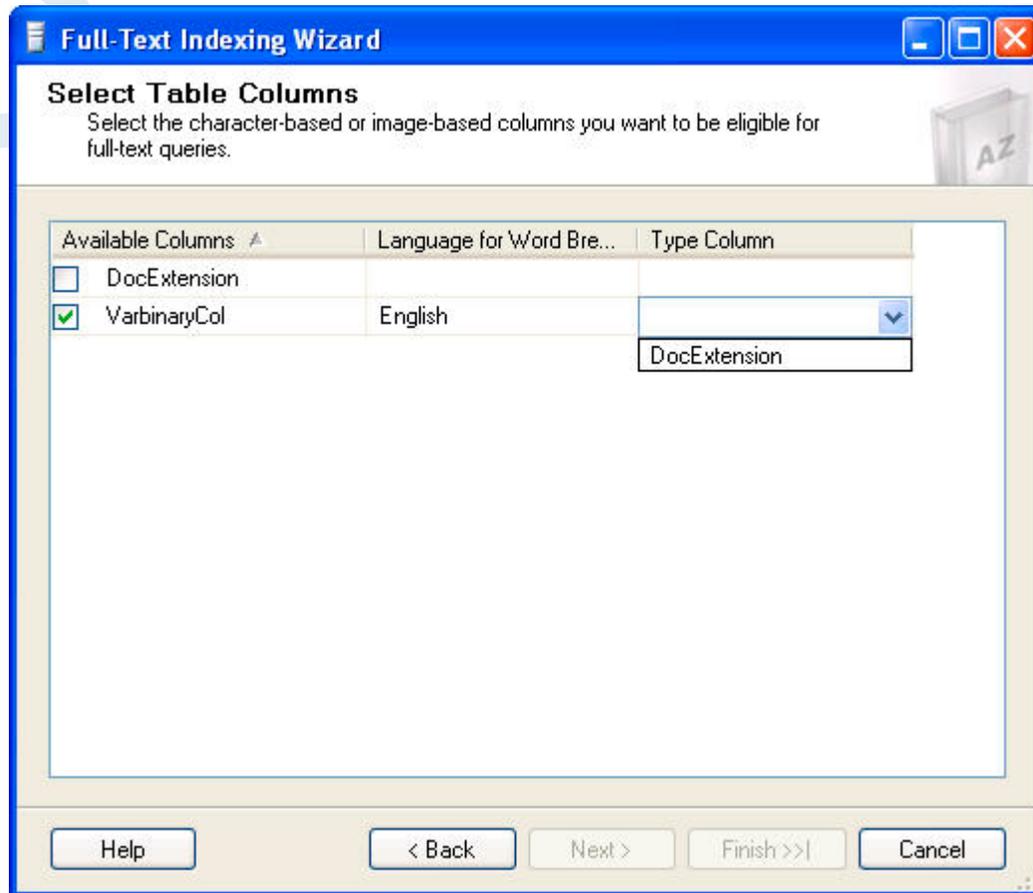
Let's look at an example. Consider this table:

```
CREATE TABLE Resumes(PK
int not null identity
constraint ResumesPK
primary key, VarbinaryCol
varbinary(max),
DocExtension varchar(5))
```

In this table, the `VarbinaryCol` holds Word and PDF documents, and the `DocExtension` column holds the document

Remember: iFilters are COM components, which the Indexer uses to extract textual data from binary file formats, such as like TIFFs, PDFs, and Word documents.

FIGURE 8: CONFIGURING THE DOCUMENT TYPE COLUMN



extension. I use the `varchar(5)` data type; it is wide enough to hold five characters, and the `varchar` datatype ensures that there is no trailing white space.

Here is the index creation statement to use:

```
CREATE FULLTEXT INDEX ON
Resumes (VarBinaryCol TYPE
COLUMN DocExtension)
KEY INDEX ResumesPK ON
catalog_name
```

When you run this command, you get the following warning message in Query Analyzer:

Warning: Table 'NativeTable' does not have the option 'text in row' enabled and has full-text indexed columns that are of type image, text, or ntext.

Full-text change tracking cannot track `WRITETEXT` or `UPDATETEXT` operations performed on these columns.

Despite the ominous note of the warning, this only applies if you are inserting or updating your binary data using WRITE-TEXT or UPDATETEXT.

To full-text index `Image` or `Varbinary(MAX)` data type columns using the Full-Text Wizard, you need to specify the document extension in the Select Full-Text Table columns, as illustrated in Figure 8.

Here's one cool feature: when you click in the type column, the columns which could possibly serve as document type columns are "auto-sensed."

The rest of the wizard is identical to that of other document types.

XML IN FTS

You can create a full-text index on XML documents if they are stored in the XML data type column or in the `Image` or `Varbinary(Max)` columns. You can query the entire column

looking for XML documents which contain the search token you are looking for.

However, real life is somewhat more complex. You may issue such relational queries against XML documents, and at the same time incorporate the hierarchical features of XML. This is important, as XML documents are often very large; they may contain many elements, and sometimes even multiple XML documents in a single column in a row!

For example, consider this XML document:

```
<document>
<book><title>
Search Me: Using SQL Server
Full-Text Search
</title>
<author title="Ms"
name="Hilary Cotter"/>
<chapter>
<title>XML</title>
```

```
</chapter>
</book>
</document>
```

If you were to search `XML` you would find this document in the chapter title. But suppose you wanted to search on rows where the chapter title element contained `XML`?

A simple contains query like this one:

```
Select * from XMLTable
where Contains(*,'XML')
```

would return rows which contain the above document, but also this one:

```
<document>
<book><title>
The Essential XML
</title>
<author title="Ms"
name="Aaron Skonnard"/>
<chapter>
<title>ADO.Net</title>
```

```
</chapter>
</book>
</document>
```

The second document shows up in the search results because the **Contains** operator can't distinguish in which XML element the search term occurs in the XML document. To restrict which element contains the search term, you have to combine SQL FTS **Contains** predicates with XPATH queries. To do this:

- 1 Store your XML documents in columns of the XML data type
- 2 Create an XML Schema on your XML data type column

Let's look at an example. This example consists of several parts, in part A we are querying the table in relationally. In parts B and C we combine SQL FTS with XPATH queries which allow us to incorporate the hierarchical aspects of XML in our full-text queries.

```
create database FullTextXML
GO
use FullTextXML
GO
--Creating our schema
CREATE XML SCHEMA COLLECTION PlaysSchema AS '
<xsd:schema targetNamespace="http://www.plays.com/plays"
  xmlns ="http://www.plays.com/plays"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
<xsd:element name="plays"><xsd:complexType><xsd:
  complexContent>
<xsd:restriction base="xsd:anyType"><xsd:choice
  maxOccurs="unbounded"><xsd:element name="folder">
<xsd:complexType><xsd:complexContent><xsd:restriction
  base="xsd:anyType"><xsd:sequence>
<xsd:element name="page" minOccurs="0"
  maxOccurs="unbounded"><xsd:complexType>
<xsd:complexContent><xsd:restriction base="xsd:anyType"><xsd:
  sequence>
```

```

<xsd:element name="submitdate" type="xsd:
string" minOccurs="0" />
<xsd:element name="pagetext" type="xsd:
string" minOccurs="0" />
</xsd:sequence><xsd:attribute name="id"
type="xsd:string"
/></xsd:restriction></xsd:complexContent>
</xsd:complexType></xsd:element></xsd:
sequence><xsd:attribute name="id"
type="xsd:string" />
</xsd:restriction></xsd:complexContent></xsd:
complexType></xsd:element></xs&Aacute:choice></
xsd:restriction>
</xsd:complexContent></xsd:complexType></
xsd:element></xsd:schema>'

```

GO

--Creating our table

```

Create Table XMLFULLText (pk int not null
identity constraint XMLFULLTEXTPK
primary key, XMLDOC XML(PlaysSchema))

```

GO

--inserting a document

--here's our document

```

declare @XMLDOC XML
set @XMLDOC ='<plays xmlns="http://www.
plays.com/plays"> <folder id="681"><page id=
"3155"> <submitdate>12-10-02</
submitdate>
<pagetext>Horatio says "tis but our fantasy,</
pagetext>
</page>
<page id="9267">
<submitdate>09-04-04</submitdate>
<pagetext>And will not let belief take hold of
him</pagetext>
</page>
</folder>
<folder id="902">
<page id="1853">
<submitdate>22-11-05</submitdate>
<pagetext>Touching this dreaded sight, twice
seen of
us:</pagetext>
</page>
<page id="8423">

```

```
<submitdate>31-05-02</submitdate>
<pagetext>Therefore I have entreated him along</
pagetext>
</page>
</folder></plays>'
--here's the insert
insert into XMLFULLText (XMLDOC) values(@
XMLDOC)
GO
--creating our full-text index
sp_fulltext_database 'enable'
GO
create fulltext catalog test as default
GO
create fulltext index on XMLFULLText (XMLDOC)
key index XMLFULLTextPK
GO
--PART A - searching relationally using SQL FTS
select * from XMLFULLText where
contains(*,'belief')
GO
--1 row returned.
```

```
--OK, let's make this a little more interesting
GO
--adding another document – this time belief
occurs in a
--different element
declare @XMLDOC XML
set @XMLDOC ='<plays xmlns="http://www.plays.
com/plays"> <folder id="681">
<page id="3155">
<submitdate>12-10-02</submitdate>
<pagetext>Horatio says "tis but our
fantasy,</pagetext>
</page>
<page id="9267">
<submitdate>09-04-04</submitdate>
<pagetext>And will not let beliefs take hold of
him</pagetext>
</page>
</folder>
<folder id="902">
<page id="1853">
<submitdate>22-11-05</submitdate>
```

```

<pagetext>Touching this dreaded sight, twice
seen of
us:</pagetext>
</page>
<page id="8423">
<submitdate>31-05-02</submitdate>
<pagetext>Therefore I have entreated him
along</pagetext>
</page>
</folder>
</plays>'
--inserting a new document
insert into XMLFULLText (XMLDOC) values(@
XMLDOC)
GO
select * from XMLFULLText where
CONTAINS(*,'belief')
GO
--1 row returned.
select * from XMLFULLText where
FREETEXT(*,'belief')
GO

```

```

--2 row returned. lement
--PART B - searching combining Full-Text with
XPath
--combining SQL FTS queries with xpath to
search only the submitdate element
WITH XMLNAMESPACES ('http://www.plays.
com/plays' AS pd)
SELECT pk,*
FROM XMLFulltext
where contains(XMLDOC, 'belief')
and
xmldoc.exist('/pd:plays[1]/pd:folder[1]/pd:
page[1]/pd:submitdate[contains(.,"belief")]')=1
GO
--Part C - combining SQL FTS queries with xpath
to search only the pagetext element
WITH XMLNAMESPACES ('http://www.plays.
com/plays' AS pd)
SELECT pk,*
FROM XMLFulltext
where contains(XMLDOC, 'belief')
and
xmldoc.exist('/pd:plays[1]/pd:folder[1]/pd:

```

```
page[2]/pd:pagetext[contains(.,
"belief"))]=1
```

GO

NOISE WORDS AND THE THESAURUS OPTION

Noise words are words which occur frequently in text but are not useful in searches. These include personal pronouns, such as *my*, *her*, *his*; articles *the*, *is*, *at*; and also words specific to a business, such as the company name. Consider searching on the Microsoft Web site for *Microsoft*; *Microsoft* is on each page at least once. If you were designing Microsoft's databases, the word *Microsoft* would not help anyone find useful data, so it is considered to be noise. In another context, of course — such as your own company's accounts payables database — it certainly would have relevance.

You can add or remove noise words from the default noise word lists, which are stored in a group of files in the directory

C:\Program Files\Microsoft SQL Server\MSSQL.X\MSSQL\FTData.

Table 2 illustrates the noise word file and which language applies to it.

If you search on a word which is in your noise word list you get the following error:

```
Informational: The full-text search condition contained noise word(s).
```

```
pk      charcoal
```

```
-----
(0 row(s) affected)
```

You can disable this behavior by clearing the noise word list and rebuilding your catalog, or by issuing the following commands:

```
sp_configure 'show advanced options', 1
```

```
RECONFIGURE
```

GO

```
sp_configure 'transform noise words', 1
```

TABLE 2: NOISE WORD FILES AND THEIR RESPECTIVE LANGUAGES

FILENAME	LANGUAGE
NoiseCHS.txt	Chinese Simplified
NoiseCHT.txt	Chinese Traditional
noiseDAN.txt	Danish
noiseDEU.txt	German
noiseENG.txt	International English
noiseENU.txt	United States English
noiseESN.txt	Spanish
noiseFRA.txt	French
noiseITA.txt	Italian
noiseJPN.txt	Japanese
noiseKOR.txt	Korean
noiseNEU.txt	Neutral
noiseNLD.txt	Dutch
noisePLK.txt	Polish
noisePTB.txt	Portuguese - Brazilian
noisePTS.txt	Portuguese - Iberian
noiseRUS.txt	Russian
noiseSVE.txt	Swedish
noiseTHA.txt	Thai
noiseTRK.txt	Turkish

RECONFIGURE

GO

If a word occurs too frequently in your text and is not helpful in resolving queries (as with our example of *Microsoft* on the *Microsoft* Web site), you can add it to your noise word list, and it will no longer be indexed. You must restart MSFTESQL before the noise word changes take effect.

Another option you have in word control is the thesaurus. The thesaurus option allows you to expand your search term to synonyms (words which have similar meanings to another word). For example, you could use the thesaurus option to indicate that a search on *color* should also search on *colour*. This is termed an *expansion*.

You can also replace a word using the replacement feature; such as when you want the word *sex* to be replaced by a search for *gender*.

The thesaurus option is automatic using the `FreeText` predicate, but you can also use the `Contains` predicate with the `FormsOf` operator, like this:

```
Select * from Mytable where
Contains(*,formsOf(thesaurus,"test"))
```

The thesaurus files are located in `C:\Program Files\Microsoft SQL Server\MSSQL.X\MSSQL\FTdata`. Table 3 shows a list of each language file.

A thesaurus file stores the data in XML, and looks like this:

```
<XML ID="Microsoft Search
Thesaurus">
<!-- Commented out
<thesaurus xmlns="x-
schema:tsSchema.xml">
<diacritics_sensitive>0</
diacritics_sensitive>
<expansion>
<sub>Internet Explor-
er</sub>
<sub>IE</sub>
<sub>IE5</sub>
```

TABLE 3: THESAURUS FILES NAMES AND THEIR LANGUAGES

FILENAME	LANGUAGE
tsCHS.xml	Chinese Simplified
tsCHT.xml	Chinese Traditional
tsDAN.xml	Danish
tsDEU.xml	German
tsENG.xml	International English
tsENU.xml	United States English
tsESN.xml	Spanish
tsFRA.xml	French
tsGLOBAL.xml	Global: entries here are effective in all languages
tsITA.xml	Italian
tsJPN.xml	Japanese
tsKOR.xml	Korean
tsNLD.xml	Dutch
tsPLK.xml	Polish
tsPTB.xml	Portuguese - Brazilian
tsPTS.xml	Portuguese - Iberian
tsRUS.xml	Russian
tsSVE.xml	Swedish
tsTHA.xml	Thai
tsTRK.xml	Turkish

```

</expansion>
<replacement>
  <pat>NT5</pat>
  <pat>W2K</pat>
  <sub>Windows 2000</
sub>
</replacement>
<expansion>
  <sub>run</sub>
  <sub>jog</sub>
</expansion>
</thesaurus>
-->
</XML>:

```

To use the thesaurus option, remove the comment tags. In this example, Internet Explorer has the expansions to IE and IE5. A search on *Internet Explorer* will return Internet Explorer, *IE* and *IE5*. A search on *IE* returns results that include *IE*, *Internet Explorer*, and *IE5*. You can define your own expansions by wrapping substitutions in an expansion tag. Wrap substitutions in a **sub** tag.

To create a replacement,

wrap the replacement in a replacement tag, each pattern you want replaced by a **pat** tag, and then wrap the replacement with a **sub** tag.

Note the diacritic sensitive tag (`<diacritics_sensitive>0</diacritics_sensitive>`). Setting this to a value of 1 tells SQL Server to be insensitive to accents when applying the thesaurus. Assuming that you want *café* replaced by *coffee*, your replacement tag looks like this:

```

<replacement>
  <pat>café</pat>
  <sub>coffee</sub>
</replacement>

```

If the `diacritics_sensitive` setting is 0, the word *cafe* would not be replaced by *coffee*; but *café* would be.

TESTING FULL-TEXT SEARCH IMPLEMENTATIONS

Once you create your full-text index and populate your catalogs, you can test your solution by issuing full-text queries.

By now, you should be familiar with the basic types of queries. Here is an example:

```

Select * from TableName
where contains(*,'test')

```

Where you know the word “test” is in your content. Further coverage of the search SQL syntax will be covered in another ebook.

SEARCH

If the above query does not return any results or it give you results you didn't expect, check the following:

Ensure full text is installed. Issue the below query:

```

SELECT fulltextserviceproper
ty('IsFulltextInstalled')

```

A value of 1 means it is installed.

Then check the status of your population. First, verify your catalog name. The easiest way to do this is to issue this query to determine the catalogs are in your database:

```
sp_help_fulltext_catalogs
```

and then check to see which table your catalog is in. To do this, issue this query:

```
sp_help_fulltext_tables
```

Next, check the catalog population state and finally the status of the population of your



The log file is plain text and can be read by any text editor. Use the error messages logged in that file to help you diagnose the problem.

table. Use the following commands to do this:

```
--the number of documents
in the catalog CatalogName
select FULLTEXTCATALOG-
PROPERTY(' CatalogName','it
emcount')
--the population status of
the catalog CatalogName
select FULLTEXTCATALOG-
PROPERTY(' CatalogName','P
opulateStatus')
--the number of unique
words indexes in the catalog
CatalogName
select FULLTEXTCATALOG-
PROPERTY(' CatalogName','U
niqueKeyCount')
--the number of rows pro-
cessed in the table you are
full-text indexing. In this
case the table is called full-
text
select
OBJECTPROPERTYEX(object_i
d('fulltext'),'TableFulltextDoc
sProcessed')
```

```
--the status of the full-text
population of the table full-
text
```

```
select
OBJECTPROPERTYEX(object_i
d('fulltext'),'TableFulltextPop
ulateStatus')
```

SQL FTS keeps a log of the population in C:\Program Files\Microsoft SQL Server\MSSQL.X\MSSQL\Log. The log naming convention is +db_id()+0000X.log.Y where db_id() is your database id. X is an identifier corresponding to the number of full-text catalogs in your database, and Y is the log version number.

The log file is plain text and can be read by any text editor. Use the error messages logged in that file to help you diagnose the problem.

OPTIMIZING FULL-TEXT SEARCH

SQL FTS offers orders of magnitude performance advantages

querying over the `like` operator. However, it does cause some performance degradation, especially while indexing, and can cause locking on live tables. Querying also can be problematic, and we have covered some of the techniques you can use to offer better querying performance above.

The bulk of the next section covers how to improve indexing speed so that you can deliver a search solution which has minimal performance impact on your SQL Server, and everyone

comes running and say “Wow, performance is great, what did you do?”

There are several factors to deliver an optimal full-text search solution.

ARCHITECTURE

Avoid using the `Image` or `Varbinary` data type columns for your content; binary content indexes slower than does pure text content. When possible, convert all content to `text`, and store the data in the `varchar(max)` data type columns.

If you have a large table, break the table into partitions, and dedicate a separate catalog to each partition. With partitioned tables (not to be confused with table partitioning) you can now issue full-text queries across linked servers. That is, if you issue a full-text query against a remote server, the processing is performed on that remote server, and then the

results are sent back to the calling server.

SQL FTS does not take advantages of table partitioning as the full-text queries do not align themselves with the table partitions, but if you do break large monolithic tables into smaller tables, you will notice radical performance increase in your full-text querying. Another advantage is that if only one of your partition tables is live (i.e. undergoing inserts, update, and deletes), the other partitions can be set with change tracking off, which frees up more resources for your SQL Server.

HARDWARE

Use the fastest disk subsystem for your catalogs. Ideally, use a RAID 10 array on its own controller. SQL FTS benefits from multiple processors, preferably four or more. 64-bit hardware also offers substantial performance benefits.

“When possible, convert all content to text, and store the data in the `varchar(max)` data type columns.”

ENVIRONMENT

Here are some environment tips that you can use to configure your SQL Server or optimal performance:

- ✦ **Dedicate a least 512MB to 1GB of RAM to MSFT-ESQL by setting max server memory to 1GB less than installed memory.**
- ✦ **Set Resource Usage to run at 5 to give a performance boost to the indexing process (`sp_full-text_service 'resource_usage',5`).**
- ✦ **Set ft crawl bandwidth (max) and ft notify bandwidth (max) to 0.**
- ✦ **Set max full-text crawl range to the number of CPUs on your system.**

Use `sp_configure` to perform these changes.

BACKING UP SQL FTS CATALOGS

As mentioned previously, in

SQL 2005, full-text catalogs are stored with a backup and with the database files if you detach them. This offers considerable advantages, as building the catalogs can be a time-consuming processes.

In SQL 2000, the catalogs remain in the file system. This Microsoft Knowledgebase article details the process necessary to backup and restore your full-text catalogs <http://support.microsoft.com/kb/240867/>

SUMMARY

I hope you leave this article with a good understanding of SQL FTS. It's a hugely powerful feature that can improve both the performance of your databases and the quality of results for the end-user.

Central to your understanding should be that SQL FTS builds an index just like the one you find at the back of most books, so you can find all rows which

contain a word or a term. You should also know, by now, when and where to use change tracking (use it all the time, except when the bulk of your content changes at regular intervals), over full or incremental populations. You should also know that when your search solution gets big, it's time to limit your results set and to think about partitioning your tables.

With the Internet we now have global markets, so it's more important than ever to find better ways to communicate and to find information. Most importantly, you should be able to deliver an optimal search solution that offers extremely fast performance.

Happy searching, I hope you find what you are looking for!

ABOUT THE AUTHOR

Hilary Cotter has been involved in IT for more than 20 years as a Web and database consultant. Microsoft first awarded Cotter the Microsoft SQL Server MVP award in 2001. Cotter received his bachelor of applied science degree in mechanical engineering from the University of Toronto and studied economics at the University of Calgary and computer science at UC Berkeley. He is the author of a book on SQL Server transactional replication and is currently working on books on merge replication and Microsoft search technologies.

**Remember these guidelines when using SQL FTS:**

- Always use change tracking over full or incremental populations, except, when the bulk of your content changes at regular intervals.
- When your search solution gets big, limit your results set and partition your tables.

Enjoy your new fast performing, optimal search solution.

SQL FTS is a hugely powerful feature that can improve both the performance of your databases and the quality of results for the end-user.

Additional Resources from Dell

- ✎ **System of Excellence: IT Infrastructure Keeps the Kenton School System at the Head of the Class**
www.dell.com/sql
- ✎ **SQL Server 2005: Preparing for a Smooth Upgrade**
<http://www.dell.com/downloads/global/power/ps1q06-20060126-Microsoft.pdf>
- ✎ **Maximizing SQL Server Performance**
<http://www.dell.com/downloads/global/power/ps4q05-20050272-Symantec.pdf>
- ✎ **The Scalable Enterprise Technology Center**
<http://www.dell.com/content/topics/global.aspx/power/en/setc?c=us&cs=555&l=en&s=biz>
- ✎ **Microsoft SQL Server 2005 Virtualization**
<http://www.dell.com/downloads/global/power/ps4q06-20060405-Muirhead.pdf>
- ✎ **The Definitive Guide to Scaling Out SQL Server 2005**
http://www.dell.com/content/topics/global.aspx/alliances/en/ebook_landing?c=us&cs=555&l=en&s=biz