# Full-text Search

Full-Text Search in SQL Server and Azure SQL Database lets users and applications run full-text queries against character-based data in SQL Server tables.

A full-text index includes one or more character-based columns in a table. These columns can have any of the following data types: **char**, **varchar**, **nchar**, **nvarchar**, **text**, **ntext**, **image**, **xml**, or **varbinary(max)** and **FILESTREAM**. Each full-text index indexes one or more columns from the table, and each column can use a specific language.

Full-text queries perform linguistic searches against text data in full-text indexes by operating on words and phrases based on the rules of a particular language such as English or Japanese. Full-text queries can include simple words and phrases or multiple forms of a word or phrase. A full-text query returns any documents that contain at least one match (also known as a *hit*). A match occurs when a target document contains all the terms specified in the full-text query, and meets any other search conditions, such as the distance between the matching terms.

Full-Text Search queries

After columns have been added to a full-text index, users and applications can run full-text queries on the text in the columns. These queries can search for any of the following:

* One or more specific words or phrases (*simple term*)
* A word or a phrase where the words begin with specified text (*prefix term*)
* Inflectional forms of a specific word (*generation term*)
* A word or phrase close to another word or phrase (*proximity term*)
* Synonymous forms of a specific word (*thesaurus*)
* Words or phrases using weighted values (*weighted term*)

Full-text queries are not case-sensitive. For example, searching for "Aluminum" or "aluminum" returns the same results.

Full-text queries use a small set of Transact-SQL predicates (CONTAINS and FREETEXT) and functions (CONTAINSTABLE and FREETEXTTABLE). However, the search goals of a given business scenario influence the structure of the full-text queries. For example:

* e-business—searching for a product on a website:

SELECT product\_id FROM products

WHERE CONTAINS(product\_description, ”Snap Happy 100EZ” OR FORMSOF(THESAURUS,’Snap Happy’) OR ‘100EZ’) AND product\_cost < 200 ;

* Recruitment scenario—searching for job candidates that have experience working with SQL Server:

SELECT candidate\_name,SSN FROM candidates

WHERE CONTAINS(candidate\_resume,”SQL Server”) AND candidate\_division =DBA;

## Compare Full-Text Search queries to the LIKE predicate

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

## Full-Text Search architecture

Full-text search architecture consists of the following processes:

* The SQL Server process (sqlservr.exe).
* The filter daemon host process (fdhost.exe).

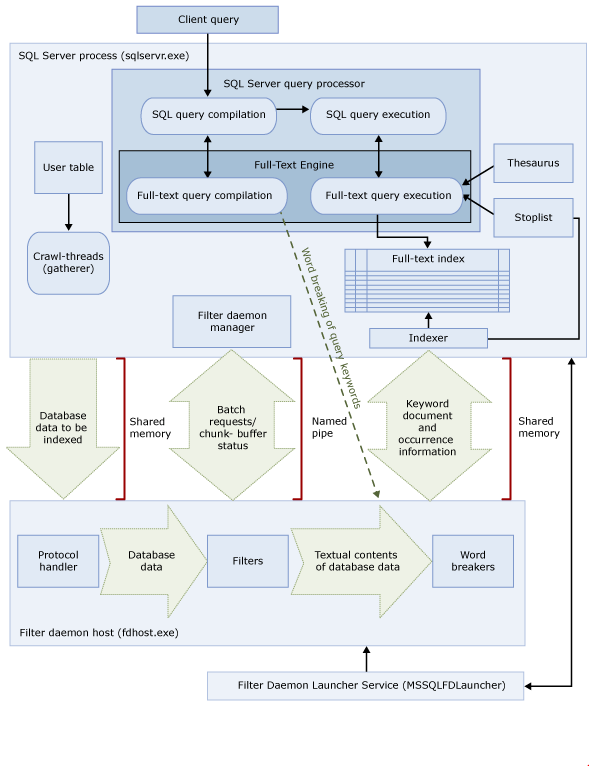
For security reasons, filters are loaded by separate processes called the filter daemon hosts. The fdhost.exe processes are created by an FDHOST launcher service (MSSQLFDLauncher), and they run under the security credentials of the FDHOST launcher service account. Therefore, the FDHOST launcher service must be running for full-text indexing and full-text querying to work.

These two processes contain the components of the full-text search architecture. These components and their relationships are summarized in the following illustration. The components are described after the illustration.

### SQL Server process

The SQL Server process uses the following components for full-text search:

* **User tables.** These tables contain the data to be full-text indexed.
* **Full-text gatherer.** The full-text gatherer works with the full-text crawl threads. It is responsible for scheduling and driving the population of full-text indexes, and also for monitoring full-text catalogs.
* **Thesaurus files.** These files contain synonyms of search terms.
* **Stoplist objects.** Stoplist objects contain a list of common words that are not useful for the search.
* **SQL Server query processor.** The query processor compiles and executes SQL queries. If a SQL query includes a full-text search query, the query is sent to the Full-Text Engine, both during compilation and during execution. The query result is matched against the full-text index.
* **Full-Text Engine.** The Full-Text Engine in SQL Server is fully integrated with the query processor. The Full-Text Engine compiles and executes full-text queries. As part of query execution, the Full-Text Engine might receive input from the thesaurus and stoplist.
* **Index writer (indexer).** The index writer builds the structure that is used to store the indexed tokens.
* **Filter daemon manager.** The filter daemon manager is responsible for monitoring the status of the Full-Text Engine filter daemon host.



### Filter Daemon Host process

The filter daemon host is a process that is started by the Full-Text Engine. It runs the following full-text search components, which are responsible for accessing, filtering, and word breaking data from tables, as well as for word breaking and stemming the query input.

The components of the filter daemon host are as follows:

* **Protocol handler.** This component pulls the data from memory for further processing and accesses data from a user table in a specified database. One of its responsibilities is to gather data from the columns being full-text indexed and pass it to the filter daemon host, which will apply filtering and word breaker as required.
* **Filters.** Some data types require filtering before the data in a document can be full-text indexed, including data in **varbinary**, **varbinary(max)**, **image**, or **xml** columns. The filter used for a given document depends on its document type. For example, different filters are used for Microsoft Word (.doc) documents, Microsoft Excel (.xls) documents, and XML (.xml) documents. Then the filter extracts chunks of text from the document, removing embedded formatting and retaining the text and, potentially, information about the position of the text. The result is a stream of textual information. For more information, see [Configure and Manage Filters for Search](https://docs.microsoft.com/en-us/sql/relational-databases/search/configure-and-manage-filters-for-search).
* **Word breakers and stemmers.** A word breaker is a language-specific component that finds word boundaries based on the lexical rules of a given language (word breaking). Each word breaker is associated with a language-specific stemmer component that conjugates verbs and performs inflectional expansions. At indexing time, the filter daemon host uses a word breaker and stemmer to perform linguistic analysis on the textual data from a given table column. The language that is associated with a table column in the full-text index determines which word breaker and stemmer are used for indexing the column.

## Full-Text Search processing

Full-text search is powered by the Full-Text Engine. The Full-Text Engine has two roles: indexing support and querying support.

### Full-Text indexing process

When a full-text population (also known as a crawl) is initiated, the Full-Text Engine pushes large batches of data into memory and notifies the filter daemon host. The host filters and word breaks the data and converts the converted data into inverted word lists. The full-text search then pulls the converted data from the word lists, processes the data to remove stopwords, and persists the word lists for a batch into one or more inverted indexes.

When indexing data stored in a **varbinary(max)** or **image** column, the filter, which implements the **IFilter** interface, extracts text based on the specified file format for that data (for example, Microsoft Word). In some cases, the filter components require the **varbinary(max)**, or **image** data to be written out to the filterdata folder, instead of being pushed into memory.

As part of processing, the gathered text data is passed through a word breaker to separate the text into individual tokens, or keywords. The language used for tokenization is specified at the column level, or can be identified within **varbinary(max)**, **image**, or **xml** data by the filter component.

Additional processing may be performed to remove stopwords, and to normalize tokens before they are stored in the full-text index or an index fragment.

When a population has completed, a final merge process is triggered that merges the index fragments together into one master full-text index. This results in improved query performance since only the master index needs to be queried rather than a number of index fragments, and better scoring statistics may be used for relevance ranking.

### Full-Text querying process

The query processor passes the full-text portions of a query to the Full-Text Engine for processing. The Full-Text Engine performs word breaking and, optionally, thesaurus expansions, stemming, and stopword (noise-word) processing. Then the full-text portions of the query are represented in the form of SQL operators, primarily as streaming table-valued functions (STVFs). During query execution, these STVFs access the inverted index to retrieve the correct results. The results are either returned to the client at this point, or they are further processed before being returned to the client.

## Full-text index architecture

The information in full-text indexes is used by the Full-Text Engine to compile full-text queries that can quickly search a table for particular words or combinations of words. A full-text index stores information about significant words and their location within one or more columns of a database table. A full-text index is a special type of token-based functional index that is built and maintained by the Full-Text Engine for SQL Server. The process of building a full-text index differs from building other types of indexes. Instead of constructing a B-tree structure based on a value stored in a particular row, the Full-Text Engine builds an inverted, stacked, compressed index structure based on individual tokens from the text being indexed. The size of a full-text index is limited only by the available memory resources of the computer on which the instance of SQL Server is running.

Beginning in SQL Server 2008, the full-text indexes are integrated with the Database Engine, instead of residing in the file system as in previous versions of SQL Server. For a new database, the full-text catalog is now a virtual object that does not belong to any filegroup; it is merely a logical concept that refers to a group of the full-text indexes. Note, however, that during upgrade of a SQL Server 2005 database, any full-text catalog that contains data files, a new filegroup is created;

Only one full-text index is allowed per table. For a full-text index to be created on a table, the table must have a single, unique nonnull column. You can build a full-text index on columns of type **char**, **varchar**, **nchar**, **nvarchar**, **text**, **ntext**, **image**, **xml**, **varbinary**, and **varbinary(max)** can be indexed for full-text search. Creating a full-text index on a column whose data type is **varbinary**, **varbinary(max)**, **image**, or **xml** requires that you specify a type column. A type column is a table column in which you store the file extension (.doc, .pdf, .xls, and so forth) of the document in each row.

### Differences between full-text indexes and regular SQL Server indexes:.

| Full-text indexes | Regular SQL Server indexes |
| --- | --- |
| Only one full-text index allowed per table. | Several regular indexes allowed per table. |
| The addition of data to full-text indexes, called a population, can be requested through either a schedule or a specific request, or can occur automatically with the addition of new data. | Updated automatically when the data upon which they are based is inserted, updated, or deleted. |
| Grouped within the same database into one or more full-text catalogs. | Not grouped. |

## Full-Text search linguistic components and language support

Full-text search supports almost 50 diverse languages, such as English, Spanish, Chinese, Japanese, Arabic, Bengali, and Hindi. For a complete list of the supported full-text languages, see [sys.fulltext\_languages (Transact-SQL)](https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-fulltext-languages-transact-sql). Each of the columns contained in the full-text index is associated with a Microsoft Windows locale identifier (LCID) that equates to a language that is supported by full-text search. For example, LCID 1033 equates to U.S English, and LCID 2057 equates to British English. For each supported full-text language, SQL Server provides linguistic components that support indexing and querying full-text data that is stored in that language.

Language-specific components include the following:

* **Word breakers and stemmers.** A word breaker finds word boundaries based on the lexical rules of a given language (word breaking). Each word breaker is associated with a stemmer that conjugates verbs for the same language. For more information, see [Configure and Manage Word Breakers and Stemmers for Search](https://docs.microsoft.com/en-us/sql/relational-databases/search/configure-and-manage-word-breakers-and-stemmers-for-search).
* **Stoplists.** A system stoplist is provided that contains a basic set stopwords (also known as noise words). A stopword is a word that does not help the search and is ignored by full-text queries. For example, for the English locale words such as "a", "and", "is", and "the" are considered stopwords. Typically, you will need to configure one or more thesaurus files and stoplists. For more information, see [Configure and Manage Stopwords and Stoplists for Full-Text Search](https://docs.microsoft.com/en-us/sql/relational-databases/search/configure-and-manage-stopwords-and-stoplists-for-full-text-search).
* **Thesaurus files.** SQL Server also installs a thesaurus file for each full-text language, as well as a global thesaurus file. The installed thesaurus files are essentially empty, but you can edit them to define synonyms for a specific language or business scenario. By developing a thesaurus tailored to your full-text data, you can effectively broaden the scope of full-text queries on that data. For more information, see [Configure and Manage Thesaurus Files for Full-Text Search](https://docs.microsoft.com/en-us/sql/relational-databases/search/configure-and-manage-thesaurus-files-for-full-text-search).
* **Filters (iFilters).** Indexing a document in a **varbinary(max)**, **image**, or **xml** data type column requires a filter to perform extra processing. The filter must be specific to the document type (.doc, .pdf, .xls, .xml, and so forth). For more information, see [Configure and Manage Filters for Search](https://docs.microsoft.com/en-us/sql/relational-databases/search/configure-and-manage-filters-for-search).

Word breakers (and stemmers) and filters run in the filter daemon host process (fdhost.exe).

Get Started with Full-Text Search

SQL Server databases are full-text enabled by default. Before you can run full-text queries, however, you must create a full text catalog and create a full-text index on the tables or indexed views you want to search.

## Set up full-text search in two steps

There are two basic steps to set up full-text search:

1. Create a full-text catalog.
2. Create a full-text index on tables or indexed view you want to search.

Each full-text index must belong to a full-text catalog. You can create a separate text catalog for each full-text index, or you can associate multiple full-text indexes with a given catalog. A full-text catalog is a virtual object and does not belong to any filegroup. The catalog is a logical concept that refers to a group of full-text indexes.

CREATE FULLTEXT CATALOG AdvWksDocFTCat;

CREATE UNIQUE INDEX ui\_ukDoc ON Production.Document(DocumentID);

CREATE FULLTEXT INDEX ON Production.Document

(

Document --Full-text index column name

TYPE COLUMN FileExtension --Name of column that contains file type information

Language 2057 --2057 is the LCID for British English

)

KEY INDEX ui\_ukDoc ON AdvWksDocFTCat --Unique index

WITH CHANGE\_TRACKING AUTO --Population type;

GO

## Choose options for a full-text index

### Choose a language

### Choose a filegroup

The process of building a full-text index is fairly I/O intensive. As a best practice, locate a full-text index in the database filegroup that is best for maximizing I/O performance or locate the full-text indexes in a different filegroup on another volume.

### Choose a full-text catalog

We recommend associating tables with the same update characteristics (such as small number of changes versus large number of changes, or tables that change frequently during a particular time of day) together under the same full-text catalog. By setting up full-text catalog population schedules, full-text indexes stay synchronous with the tables without adversely affecting the resource usage of the database server during periods of high database activity.

Consider the following guidelines:

* If you are indexing a table with millions of rows, assign the table to its own full-text catalog.
* Consider the amount of change occurring in the tables being full-text indexed, as well as the total number of rows. If the total number of rows being changed, together with the number of rows in the table present during the last full-text population, represents millions of rows, assign the table to its own full-text catalog.

### Associate a unique index

Always select the smallest unique index available for your full-text unique key. (A 4-byte, integer-based index is optimal.) This significantly reduces the resources required by Microsoft Search service in the file system. If the primary key is large (over 100 bytes), consider choosing another unique index in the table (or creating another unique index) as the full-text unique key. Otherwise, if the full-text unique key size exceeds the maximum size allowed (900 bytes), full-text population will not be able to proceed.

### Associate a stoplist

### A stoplist is a list of stopwords, also known as noise words. A stoplist is associated with each full-text index, and the words in that stoplist are applied to full-text queries on that index. By default, the system stoplist is associated with a new full-text index. You can create and use your own stoplist too

### CREATE FULLTEXT STOPLIST myStoplist FROM SYSTEM STOPLIST;

### ALTER FULLTEXT STOPLIST myStoplist ADD 'en' LANGUAGE 'Spanish';

## Update a full-text index

Like regular SQL Server indexes, full-text indexes can be automatically updated as data is modified in the associated tables. This is the default behavior. Alternatively, you can keep your full-text indexes up-to-date manually, or at specified scheduled intervals.

Updating a full-text index immediately after each change in the base table is also resource-intensive. If this occurs, consider scheduling manual change tracking updates to keep up with the numerous changes from time to time, rather than competing with queries for resources.

# Query with Full-Text Search

Write full-text queries by using the full-text predicates **CONTAINS** and **FREETEXT** and the rowset-valued functions **CONTAINSTABLE** and **FREETEXTTABLE** with the **SELECT** statement.

* Use **CONTAINS** and **CONTAINSTABLE** to match words and phrases.
* Use **FREETEXT** and **FREETEXTTABLE** to match the meaning, but not the exact wording.

## Simple examples of each predicate and function

SELECT Name, ListPrice FROM Production.Product

WHERE ListPrice = 80.99 AND CONTAINS(Name, 'Mountain')

The following example searches for all documents that contain words related to vital, safety, components.

SELECT Title FROM Production.Document

WHERE FREETEXT (Document, 'vital safety components')

Search for Words Close to Another Word with NEAR

Limit Search Results with RANK

Improve the Performance of Full-Text Queries

Search Document Properties with Search Property Lists

Find Property Set GUIDs and Property Integer IDs for Search Properties

Create and Manage Full-Text Catalogs

Create and Manage Full-Text Indexes

Choose a Language When Creating a Full-Text Index

Populate Full-Text Indexes

Improve the Performance of Full-Text Indexes

Troubleshoot Full-Text Indexing

Back Up and Restore Full-Text Catalogs and Indexes

Configure and Manage Filters for Search

Configure and Manage Word Breakers and Stemmers for Search

View or Change Registered Filters and Word Breakers

Change the Word Breaker Used for US English and UK English

Revert the Word Breakers Used by Search to the Previous Version

Customize the Behavior of Word Breakers with a Custom Dictionary

Configure and Manage Stopwords and Stoplists for Full-Text Search

Configure and Manage Thesaurus Files for Full-Text Search

Manage and Monitor Full-Text Search for a Server Instance

Set the Service Account for the Full-text Filter Daemon Launcher

Upgrade Full-Text Search

Full-Text Search DDL, Functions, Stored Procedures, and Views

Use the Full-Text Indexing Wizard

Deprecated Full-Text Search Features in SQL Server 2016

Semantic Search

Install and Configure Semantic Search

Enable Semantic Search on Tables and Columns

Find Key Phrases in Documents with Semantic Search

Find Similar and Related Documents with Semantic Search

Manage and Monitor Semantic Search

Semantic Search DDL, Functions, Stored Procedures, and Views