l a *population* (also known as a *crawl*).

ɔes of population

-text index supports the following types of population:

Full population
Automatic or manual population based on **change tracking**
Incremental population based on a **timestamp**

l population

g a full population, index entries are built for all the rows of a table or indexed view. A ɔpulation of a full-text index, builds index entries for all the rows of the base table or ed view.

fault, SQL Server populates a new full-text index fully as soon as it is created.

On the one hand, a full population can consume a significant amount of resources. Therefore, when creating a full-text index during peak periods, it is often a best practice to delay the full population until an off-peak time, particularly if the base table of an full-text index is large.

On the other hand, the full-text catalog to which the index belongs is not usable until all of its full-text indexes are populated.

eate a full-text index without populating it immediately, specify the `CHANGE_TRACKING NO POPULATION` clause in the `CREATE FULLTEXT INDEX` statement. If you specify `E_TRACKING MANUAL`, the Full-Text Engine doesn't populate the new full-text index until xecute an `ALTER FULLTEXT INDEX` statement using the `START FULL POPULATION` or `START MENTAL POPULATION` clause.

mple - Create a full-text index without running a full ɔulation

ollowing example creates a full-text index on the `Production.Document` table of the ıtureWorks sample database. This example uses `WITH CHANGE_TRACKING OFF, NO ATION` to delay the initial full population.

```
:ATE UNIQUE INDEX ui_ukDoc ON Production.Document(DocumentID);
:ATE FULLTEXT CATALOG AW_Production_FTCat;
:ATE FULLTEXT INDEX ON Production.Document

 Document                          --Full-text index column name
     TYPE COLUMN FileExtension    --Name of column that contains file type
ormation
     Language 1033                 --1033 is LCID for the English language

 KEY INDEX ui_ukDoc
 ON AW_Production_FTCat
 WITH CHANGE_TRACKING OFF, NO POPULATION;
```

mple - Run a full population on a table

ollowing example runs a full population on the `Production.Document` table of the `ıtureWorks` sample database.

```
ER FULLTEXT INDEX ON Production.Document
START FULL POPULATION;
```

ɔulation based on change tracking

ɔnally, you can use change tracking to maintain a full-text index after its initial full lation. There is a small overhead associated with change tracking because SQL Server :ains a table in which it tracks changes to the base table since the last population. ı you use change tracking, SQL Server maintains a record of the rows in the base table dexed view that have been modified by updates, deletes, or inserts. Data changes : through WRITETEXT and UPDATETEXT are not reflected in the full-text index, and are icked up with change tracking.

Note

· tables containing a **timestamp** column, you can use incremental population tead of change tracking.

ı you enable change tracking during index creation, SQL Server fully populates the ʼull-text index immediately after it is created. Thereafter, changes are tracked and ıgated to the full-text index.

ıble change tracking

: are two types of change tracking:

Automatic (`CHANGE_TRACKING AUTO` option). Automatic change tracking is the default behavior.
Manual (`CHANGE_TRACKING MANUAL` option).

ype of change tracking determines how the full-text index is populated, as follows:

Automatic population

By default, or if you specify `CHANGE_TRACKING AUTO`, the Full-Text Engine uses automatic population on the full-text index. After the initial full population completes, changes are tracked as data is modified in the base table, and the tracked changes are propagated automatically. The full-text index is updated in the background, however, so propagated changes might not be reflected immediately in the index.

To start tracking changes with automatic population

- CREATE FULLTEXT INDEX ... WITH CHANGE_TRACKING AUTO
- ALTER FULLTEXT INDEX ... SET CHANGE_TRACKING AUTO

Example - Alter a full-text index to use automatic change tracking
The following example changes the full-text index of the `HumanResources.JobCandidate` table of the `AdventureWorks` sample database to use change tracking with automatic population.

SQL Copy

```sql
USE AdventureWorks;
GO
ALTER FULLTEXT INDEX ON HumanResources.JobCandidate SET CHANGE_TRACKING
AUTO;
GO
```

Manual population

If you specify CHANGE_TRACKING MANUAL, the Full-Text Engine uses manual population on the full-text index. After the initial full population completes, changes are tracked as data is modified in the base table. However, they are not propagated to the full-text index until you execute an ALTER FULLTEXT INDEX ... START UPDATE POPULATION statement. You can use SQL Server Agent to call this Transact-SQL statement periodically.

To start tracking changes with manual population

- CREATE FULLTEXT INDEX ... WITH CHANGE_TRACKING MANUAL
- ALTER FULLTEXT INDEX ... SET CHANGE_TRACKING MANUAL

Example - Create a full-text index with manual change tracking

The following example creates a full-text index that will use change tracking with manual population on the `HumanResources.JobCandidate` table of the `AdventureWorks` sample database.

SQL Copy

```sql
USE AdventureWorks;
GO
CREATE UNIQUE INDEX ui_ukJobCand ON
HumanResources.JobCandidate(JobCandidateID);
CREATE FULLTEXT CATALOG ft AS DEFAULT;

CREATE FULLTEXT INDEX ON HumanResources.JobCandidate(Resume)
   KEY INDEX ui_ukJobCand
   WITH CHANGE_TRACKING=MANUAL;
GO
```

Example - Run a manual population

The following example runs a manual population on the change-tracked full-text index of the `HumanResources.JobCandidate` table of the `AdventureWorks` sample database.

SQL Copy

```sql
USE AdventureWorks;
GO
ALTER FULLTEXT INDEX ON HumanResources.JobCandidate START UPDATE
POPULATION;
GO
```

able change tracking

CREATE FULLTEXT INDEX ... WITH CHANGE_TRACKING OFF

ALTER FULLTEXT INDEX ... SET CHANGE_TRACKING OFF

remental population based on a timestamp

cremental population is an alternative mechanism for manually populating a full-text . If a table experiences a high volume of inserts, using incremental population can be efficient that using manual population.

an run an incremental population for a full-text index that has CHANGE_TRACKING MANUAL or OFF.

equirement for incremental population is that the indexed table must have a column **timestamp** data type. If a **timestamp** column does not exist, incremental population ot be performed.

Server uses the **timestamp** column to identify rows that have changed since the last lation. The incremental population then updates the full-text index for rows added, ed, or modified after the last population, or while the last population was in progress. e end of a population, the Full-Text Engine records a new **timestamp** value. This value

largest **timestamp** value that SQL Gatherer has found. This value will be used when ext incremental population starts.

me cases, the request for an incremental population results in a full population.

A request for incremental population on a table without a **timestamp** column results in a full population operation.
If the first population on a full-text index is an incremental population, it indexes all rows, making it equivalent to a full population.
If any metadata that affects the full-text index for the table has changed since the last population, incremental population requests are implemented as full populations. This includes metadata changes caused by altering any column, index, or full-text index definitions.

n an incremental population

o an incremental population, execute an `ALTER FULLTEXT INDEX` statement using the

n an incremental population, execute an `ALTER FULLTEXT INDEX` statement using the `INCREMENTAL POPULATION` clause.

ate or change a schedule for incremental population

In Management Studio, in Object Explorer, expand the server.

Expand **Databases**, and then expand the database that contains the full-text index.

Expand **Tables**.

Right-click the table on which the full-text index is defined, select **Full-Text index**, and on the **Full-Text index** context menu, click **Properties**. This opens the **Full-text index Properties** dialog box.

> ⓘ **Important**
>
> If the base table or view does not contain a column of the **timestamp** data type, incremental population is not possible.

In the **Select a page** pane, select **Schedules**.

Use this page to create or manage schedules for a SQL Server Agent job that starts an incremental table population on the base table or indexed view of the full-text index.

The options are as follows:

- To **create** a new schedule, click **New**.

  This opens the **New Full-Text Indexing Table Schedule** dialog box, where you can create a schedule. To save the schedule, click **OK**.

  > ⓘ **Important**
  >
  > A SQL Server Agent job (Start Incremental Table Population on *database_name.table_name*) is associated with a new schedule after you exit the **Full-Text Index Properties** dialog box. If you create multiple schedules for the same full-text index, they all use the same job.

- To **change** an existing schedule, select the existing schedule and click **Edit**.

  This opens the **New Full-Text Indexing Table Schedule** dialog box, where you

can modify the schedule.

> **Note**
>
> For information about modifying a SQL Server Agent job, see **Modify a Job**.

- To **remove** an existing schedule, select the existing schedule and click **Delete**.

Click **OK**.

ubleshoot errors in a full-text population awl)

n an error occurs during a crawl, the Full-Text Search crawl logging facility creates and tains a crawl log, which is a plain text file. Each crawl log corresponds to a particular ext catalog. By default, crawl logs for a given instance (in this example, the default nce) are located in `%ProgramFiles%\Microsoft SQL r\MSSQL15.MSSQLSERVER\MSSQL\LOG` folder.

rawl log file follows the following naming scheme:

```
<DatabaseID><FullTextCatalogID>.LOG[<n>]
```

ariable parts of the crawl log file name are the following.

<**DatabaseID**> - The ID of a database. <**dbid**> is a five digit number with leading zeros.
<**FullTextCatalogID**> - Full-text catalog ID. <**catid**> is a five digit number with leading zeros.
<**n**> - Is an integer that indicates one or more crawl logs of the same full-text catalog exist.

xample, `SQLFT0000500008.2` is the crawl log file for a database with database ID = 5, ull-text catalog ID = 8. The 2 at the end of the file name indicates that there are two log files for this database/catalog pair.

e Also

n_fts_index_population (Transact-SQL)

tarted with Full-Text Search

e and Manage Full-Text Indexes

TE FULLTEXT INDEX (Transact-SQL)

R FULLTEXT INDEX (Transact-SQL)

---

s page helpful?

'es No
