|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FULL TEXT Index Change Tracking **Problem**  Creation and maintenance of full-text catalogs and indexes in SQL Server can be taxing on both the server and the administrator, as well as take a lot of time to populate. This can result in increased deployment times and potentially inaccurate search results. To ease the deployment process and increase the uptime of your application, you can use Change-Tracking population.  **Solution**  In the traditional methods of full-text catalog population, you can choose a full or incremental population. In full population, the catalog is "built from scratch", building an index entry for each row in the table. In incremental population, a timestamp is used to determine where to start the population. The *sysfulltextnotify* table holds entries for updated rows and is polled by the MSSearch service. You can determine when this table is polled in one of three ways:   |  |  |  | | --- | --- | --- | | Method | Description | T-SQL Statement | | Scheduled | Runs on whatever schedule you choose. Can be run via a SQL Agent job | sp\_fulltext\_table *table\_name* 'Start\_change\_tracking'   sp\_fulltext\_table *table\_name* 'update\_index' | | On demand | Runs whenever you run the statements. Changes are stored in the *sysfulltextnotify* table | sp\_fulltext\_table *table\_name* 'Start\_change\_tracking'   sp\_fulltext\_table *table\_name* 'update\_index' | | Background | Changes to the table rows are propogated when they occur | sp\_fulltext\_table *table\_name* 'Start\_change\_tracking'   sp\_fulltext\_table *table\_name* 'start\_background\_updateindex' |   The background population offers a couple of benefits:   * Decreased time in propagating changes to the full-text catalog * A wider distribution of resource usage, since changes are made when the update occurs instead of having to apply the changes all at once   Once the population method is changed to change tracking, an incremental population begins to make sure the catalog is up to date. The only downside to switching to change tracking population is that entries are no longer written to the Event Log; however, you can devise another method for comparing the entries in the table with the number of rows in the full-text catalog to ensure the process is working for you.  **Full-Text Index Population**  Creating and maintaining a full-text index involves populating the index by using a process called a *population* (also known as a *crawl*). SQL Server supports the following types of population: full population, change tracking-based automatic or manual population, and incremental timestamp-based population. http://i.msdn.microsoft.com/Global/Images/clear.gif Full Population During a full population, index entries are built for all the rows of a table or indexed view. A full population of a full-text index, builds index entries for all the rows of the base table or indexed view.  By default, SQL Server populates a new full-text index fully as soon as it is created. However, 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. However, the full-text catalog to which the index belongs is not usable until all of its full-text indexes are populated. To create a full-text index without populating it immediately, specify the CHANGE\_TRACKING OFF, NO POPULATION clause in your CREATE FULLTEXT INDEX Transact-SQL statement. SQL Server will not populate the new full-text index until you execute an ALTER FULLTEXT INDEX Transact-SQL statement using the START FULL POPULATION or START INCREMENTAL POPULATION clause. For more information, see examples "A. Creating a full-text index without running a full population" and "B. Running a full population on table," later in this topic. http://i.msdn.microsoft.com/Global/Images/clear.gif Change Tracking-Based Population Optionally, you can use change tracking to maintain a full-text index after its initial full population. There is a small overhead associated with change tracking because SQL Server maintains a table in which it tracks changes to the base table since the last population. When change tracking is used, SQL Server maintains a record of the rows in the base table or indexed 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 not picked up with change tracking.   |  | | --- | | **ms142575.note(en-us,SQL.100).gifNote:** | | For tables containing a **timestamp** column, you can use incremental populations. |   When change tracking is enabled during index creation, SQL Server fully populates the new full-text index immediately after it is created. Thereafter, changes are tracked and propagated to the full-text index. There are two types of change tracking, automatic (CHANGE\_TRACKING AUTO option) and manual (CHANGE\_TRACKING MANUAL option). Automatic change tracking is the default behavior.  The type 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 set up tracking changes with automatic population**   + [CREATE FULLTEXT INDEX](http://technet.microsoft.com/en-us/library/ms187317.aspx) … WITH CHANGE\_TRACKING AUTO   + [ALTER FULLTEXT INDEX](http://technet.microsoft.com/en-us/library/ms188359.aspx) … SET CHANGE\_TRACKING AUTO   + [How to: View or Change the Properties of a Full-Text Index (SQL Server Management Studio)](http://technet.microsoft.com/en-us/library/cc879298.aspx)   For more information, see example "E. Altering a full-text index to use automatic change tracking," later in this topic.   * Manual population   If you specify CHANGE\_TRACKING MANUAL, 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. 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](http://technet.microsoft.com/en-us/library/ms187317.aspx) … WITH CHANGE\_TRACKING MANUAL   + [ALTER FULLTEXT INDEX](http://technet.microsoft.com/en-us/library/ms188359.aspx) … SET CHANGE\_TRACKING MANUAL   + [How to: View or Change the Properties of a Full-Text Index (SQL Server Management Studio)](http://technet.microsoft.com/en-us/library/cc879298.aspx)   For more information, see examples "C. Creating a full-text index with manual change tracking" and "D. Running a manual population," later in this topic.  **To set up tracking changes with no change tracking**   * [CREATE FULLTEXT INDEX](http://technet.microsoft.com/en-us/library/ms187317.aspx) … WITH CHANGE\_TRACKING OFF * [ALTER FULLTEXT INDEX](http://technet.microsoft.com/en-us/library/ms188359.aspx) … SET CHANGE\_TRACKING OFF * [How to: View or Change the Properties of a Full-Text Index (SQL Server Management Studio)](http://technet.microsoft.com/en-us/library/cc879298.aspx)  http://i.msdn.microsoft.com/Global/Images/clear.gif Incremental Timestamp-Based Population An incremental population is an alternative mechanism for manually populating a full-text index. You can run an incremental population for a full-text index that has CHANGE\_TRACKING set to MANUAL or OFF. If the first population on a full-text index is an incremental population, it indexes all rows, making it equivalent to a full population. The requirement for incremental population is that the indexed table must have a column of the **timestamp** data type. If a **timestamp** column does not exist, incremental population cannot be performed. A request for incremental population on a table without a **timestamp** column results in a full population operation. Also, 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.  SQL Server uses the **timestamp** column to identify rows that have changed since the last population. The incremental population then updates the full-text index for rows added, deleted, or modified after the last population, or while the last population was in progress. If a table experiences a high volume of inserts, using incremental population can be more efficient that using manual population.  At the end of a population, the Full-Text Engine records a new **timestamp** value. This value is the largest **timestamp** value that SQL Gatherer has encountered. This value will be used when a subsequent incremental population starts.  To run an incremental population, execute an ALTER FULLTEXT INDEX statement using the START INCREMENTAL POPULATION clause.  **To schedule an incremental population job**   * [How to: Create or Manage a Schedule for Incremental Populations of a Full-Text Index (SQL Server Management Studio)](http://technet.microsoft.com/en-us/library/cc879305.aspx)  Here are the Microsoft examples on how to work with a fulltext index:A. Creating a full-text index without running a full population The following example creates a full-text index on the **Production.Document** table of the **AdventureWorks** sample database. This example uses WITH CHANGE\_TRACKING OFF, NO POPULATION to delay the initial full population.  CREATE UNIQUE INDEX ui\_ukDoc ON Production.Document(DocumentID);  CREATE FULLTEXT CATALOG AW\_Production\_FTCat;  CREATE FULLTEXT INDEX ON Production.Document  (  Document --Full-text index column name  TYPE COLUMN FileExtension --Name of column that contains file type information  Language 1033 --1033 is LCID for the English language  )  KEY INDEX ui\_ukDoc  ON AW\_Production\_FTCat  WITH CHANGE\_TRACKING OFF, NO POPULATION;  GO B. Running a full population on table The following example runs a full population on the **Production.Document** table of the **AdventureWorks** sample database.  ALTER FULLTEXT INDEX ON Production.Document  START FULL POPULATION; C. Creating 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.  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 D. Running 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.  USE AdventureWorks;  GO  ALTER FULLTEXT INDEX ON HumanResources.JobCandidate START UPDATE POPULATION;  GO E. Altering 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.  [Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl06');" \o "Copy Code)  USE AdventureWorks;  GO  ALTER FULLTEXT INDEX ON HumanResources.JobCandidate SET CHANGE\_TRACKING AUTO;  GO |