Microsoft SQL Server Full-Text Search

James R. Hamilton
Microsoft Corporation
JamesRH@microsoft.com

Tapas K. Nayak
Microsoft Corporation
TapasNay@microsoft.com

1 Introduction

Over the last decade, the focus of the commercial database management community has been primarily on structured data and the industry as a whole has been fairly effective at addressing the needs of these structured storage applications. However, only a small fraction of the data stored and managed each year is fully structured while the vast preponderance of stored data is either wholly unstructured or only semi-structured in the form of documents, web-pages, spreadsheets, email, and other weakly structured formats.

This work investigates the features and capabilities of the full text search access method in Microsoft SQL Server 2000 and the follow-on release of this product and how these search capabilities are integrated into the query language. We will first outline the architecture for the full text search support, then describe the full text query features in more detail, and finally show examples of how this support allows single SQL queries over structured, unstructured, and semi-structured data.

2 SQL Server Search Architecture

The SQL Server full text search feature leverages the same underlying full text search access method and infrastructure employed in other Microsoft products, including Exchange, Sharepoint Portal Server, and the Indexing Service that supports full text search over filesystem hosted data. This approach has several advantages, the most significant of which are 1) common full text search semantics across data stored in relational tables, the mail system, web hosted data, and filesystem resident data, and 2) leverage of full text search access method and infrastructure investments across many complementary products.

Indexed text in SQL Server can range from a simple character string data to documents of many types, including Word, Powerpoint, PDF, Excel, HTML, and XML. The document filter support is a public interface, allowing for custom document formats to be integrated into SQL Server full text search. The architecture is composed of five modules hosted in three address spaces (see Figure 1): 1) content reader, 2) filter daemon, 3) word breaker, 4) indexer, and 5) query processor.

Full text indexed data stored in SQL Server tables is scanned by the content reader which assemble data and related metadata packets. These packets flow to the main search engine, which triggers the search engine filter daemon process to consume the data read by the content reader. Filter daemons are modules managed by MS Search but outside of the MS Search address space. Since the search architecture is extensible and filters may be sourced from the shipped product, ISV supplied or ISV produced and there is a risk that a filter bug or a

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

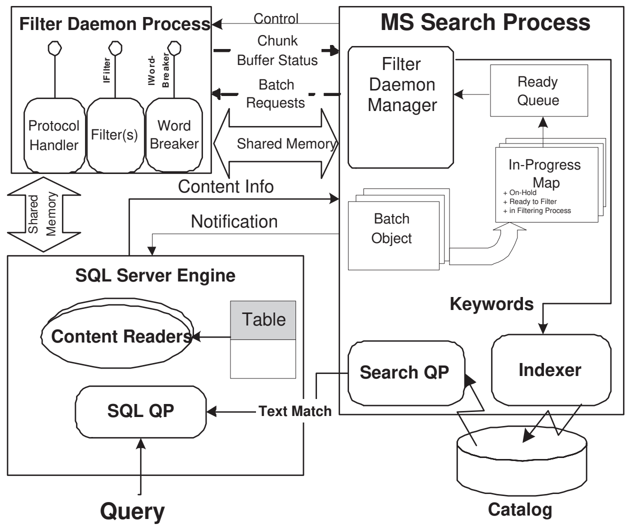

Figure 1: Architecture of SQL Server Full-Text Search.

combination of a poorly formed document and a filter bug could allow the filter to either fail or not terminate. Running the filters and word breakers in an independent process allows the system to be robust in the presence of these potential failure modes. If some instance of daemon fails or runs rogue, MS Search process restarts a new instance.

Filters are invoked by the daemon based on the type of the content. Filters parse the content and emit chunks of processed text. A chunk is a contiguous portion of text along with some relevant information about the text segment like the language-id of the text, attribute information if any, etc. Filters emit chunks separately for any properties in the content. Properties can be items such as title or author and are specific to the content types.

Word breakers break these chunks into keywords. Word breakers are modules that are human language-aware. SQL Server search installs word breakers for various languages, including but not limited to English (USA and UK), Japanese, German, French, Korean, Simplified and Traditional Chinese, Spanish, Thai, Dutch, Italian, and Swedish. The word breakers are also hosted by the filter daemons and they emit keywords, alternate keywords, and location of the keyword in the text. These keywords and related metadata are transferred to the MS Search process via a high speed shared memory protocol that feeds the data into the Indexer. The indexer builds an inverted keyword list with a batch consisting of all the keywords from one or more content items. Once MS Search persists this inverted list to disk, it sends notification back to the SQL Server process confirming success. This protocol ensures that, although documents are not synchronously indexed, documents will not be lost in the event of process or server failures and it allows the indexing process to be restartable based upon metadata maintained by the SQL Server kernel.

As with all text indexing systems we have worked upon, the indexes are stored in a highly compressed form that increases storage efficiency at the risk of driving up the cost of update. To obtain this storage size reduction without substantially increasing update cost, a stack of indexes are maintained. New documents are built into a small index, which is periodically batch merged into a larger index that, in turn, is periodically merged into the base index. This stack of indexes may be greater than three deep but the principle remains the same and it is an engineering approach that allows the use of an aggressively compressed index form without driving up the insertion costs dramatically. When searching for a keyword, all indexes in this stack need to be searched so there is some advantage in keeping the number of indexes to a small number. During insertion and merge operations, distribution and frequency statistics are maintained for internal query processing use and for ranking purposes.

This whole cycle sets up a pipeline involving the SQL Server kernel, the MS Search Engine and the filter daemons, combination of which is the key to reliability and performance of SQL Server full text indexing process.

3 SQL Server Full-text Search Query Features

The full text indexes supported by SQL Server are created using the familiar *CREATE INDEX* SQL DDL statement. These indexes are fully supported by SQL Server standard utilities, such as backup and restore, and other administrative operations, such as database attach/detach work unchanged in the presence of full text search indexes. Other enterprise-level features, including shared disk cluster failover, are fully supported in the presence of full text indexes. Indexes are created and maintained online using one of three options: *1) Full Crawl* scans the full table and builds or rebuilds a complete full text index on the indexed columns of the table. This operation proceeds online with utility progress reporting. *2) Incremental Crawl* uses a timestamp column on the indexed table to track changes to the indexed content since the last re-index. *3) Change Tracking* is used to maintain near real time currency between the full text index and the underlying text data. The SQL Server Query Processor directly tracks changes to the indexed data and this data is applied in near real time to the full text index.

The full text search support is exposed in SQL using the following constructs:

1. **Contains Predicate:** *Contains(col_list, '<search condition>')*. A contains predicate is true if any of the indicated columns in the list *col_list* contains terms that satisfy the given search condition. A search condition can be a keyword, a keyword prefix, a proximity term, or some combination of these. For example a predicate *Contains(description, ('word* or Excel or "Microsoft Access"'))* will match all entries with description containing words like 'word', 'wordings', 'Excel' or the phrase 'Microsoft Access'.

2. **Freetext Predicate:** Freetext predicates match on text containing terms that are linguistically similar (stemming) to the terms in the search condition. Thus *Freetext(description, 'run in the rain')* will match all the items that contain in its description column text with terms like *run*, *running*, *rain*, *rains*, etc.

3. **ContainsTable and FreetextTable:** ContainsTable and FreetextTable are table-valued functions that locate entries using a search condition as above, and return the matching items along with a rank value for each item computed based on term statistics in the item as well as in the whole corpus.

The search condition for any of the predicates described above can include:

1. Keyword, phrase, prefix: E.g., *Excel, Microsoft Word, word**.

2. Linguistic generation of relevant keywords: Thesaurus and Inflectional Forms: *Contains(*, 'FORMSOF(INFLECTIONAL, distributed) AND FORMSOF(THESAURUS,databases)')* will find documents containing inflectional forms of *distributed* and all words meaning the same as *databases* (thesaurus support).

3. Weighted Terms: Query terms can be assigned relative weight to impact the rank of matching documents. In the following, the *spread* search term is given twice the weight of the *sauces* search term:

```
SELECT a.CategoryName, a.Description, b.rank
FROM Categories a, ContainsTable( Categories, description,
   'ISABOUT(spread weight (.8), sauces weight(.4),
   relishes weight(.2))') b
WHERE a.categoryId = b.[key]
```

4. Proximity: One can specify queries using proximity (NEAR) between terms in a matching document, e.g., 'distributed NEAR databases' matches items in which the term *distributed* appears close to *databases*.

5. Composition: Terms A and B can be composed as A AND B, A OR B and A AND NOT B.

4 Examples of Full-text Query Scenarios

Example 1. We have a table *Documents(DocId, Title, Author, Content)* of documents published in a site. This following query finds all documents authored by Linda Chapman on *child development* and *insomnia* which include the term *child* close to the term *development* in Title. The result is presented in descending order of rank.

```
 SELECT a.Title,b.rank
 FROM Documents a,
    FreetextTable(Documents, Content, '"child development" AND insomnia') b
 WHERE a.DocId=b.[key] and a.Author='Linda Chapman' and
      Contains(a.Title, 'child NEAR development')
 ORDER BY b.rank DESC
```

Example 2 (Heterogeneous sources). Data is stored as emails (data source *exchange*), locally-authored documents (data source *monarch*), documents published in SQL Server (schema same as above). The following query gets all documents related to *marketing* and *cosmetics* from all three stores.

```
--Get qualifying email docs
  SELECT DisplayName, hRef FROM openquery(exchange,
      'SELECT "DAV:displayname" as DisplayName, "DAV:href" as hRef
       FROM "manager\Inbox"
       WHERE contains(*, ''marketing AND cosmetics'')')
UNION ALL  --Get qualifying filesystem data
  SELECT filename, vpath  FROM OpenQuery(Monarch,
      'SELECT vpath, Filename
       FROM SCOPE(''deep traversal of "c:\My Documents"'')
       WHERE contains(''marketing AND cosmetics'')')
UNION ALL  --Get qualifying SQL Server data
  SELECT Title, 'SQLServer:Documents:'+cast(DocId as varchar(9)) as href
  FROM Documents
  WHERE contains(*,'marketing AND cosmetics')
```

5 Conclusions

In this paper we motivate the integration of a native full text search access method into the Microsoft SQL Server product, describe the architecture of the access method and motivate some of the trade-offs and advantages of the engineering approach taken. We explore the features and functions of the full text search feature and provide example SQL queries showing query integration over structured, semi-structured, and unstructured data.

For further SQL Server 2000 full text search usage and feature details one may look at Inside Microsoft SQL Server [1] or SQL Server 2000 Books online [2]. On the implementation side, we are just completing a major architectural overhaul of the indexing engine and its integration with SQL Server in the next release of the product and this paper is the first description of this work.

References

[1] Delaney, Kalen. *Inside Microsoft SQL Server 2000*, Microsoft Press, 2001.

[2] *SQL Server 2000 Books Online.*