Lightning Fast Searches With FULLTEXT Indices

by Jeff Ash

Today's businesses require comprehensive searching capabilities that provide users with fast and flexible access to their data. While standard SQL provides the foundation for powerful filtering, sorting, and grouping, there are some limitations when it comes to certain types of searches. The most notable of these limitations include complex search criteria and slow performance locating data in the middle of records in large tables. Although SQL provides LIKE and REGEX for powerfully filtering data, in BBj® 15.00, BASIS added a game changer with FULLTEXT indices.

How Are FULLTEXT Indices Different?

When it comes to optimization, standard SQL filtering using <, >, =, and LIKE use the keys on the underlying data files whenever possible to reduce the number of records the query actually examines. In order to use a key, the SQL engine must be able to find matching data from the beginning of the column value. If the filter calls for examining only information from the middle of records (which results in searching all text in a column), the SQL engine cannot optimize this type of query and thus must examine all records in the entire file to find matching results. On a large table this can make the query unusably slow.

In contrast, FULLTEXT indices utilize a completely different indexing mechanism that not only allows for quickly locating data regardless of its location in the record, but additional robust filtering capabilities that extend the power and flexibility of SQL. This mechanism utilizes the best-in-class Lucene search engine that provides the ability to query data using either Google-like simplicity, or its advanced query language.

FULLTEXT Index Structure

One of the outstanding benefits of FULLTEXT indices in BBj is that they do not require any modification of the existing data file. BBj FULLTEXT indices consist of two parts: (1) a synchronization file (.sync) to tell BBj how to keep the index in synch with the data file and (2) a Lucene index (.textsearch directory). When creating a FULLTEXT index on a table, BBj creates each of these components in the same directory as the table's data file, which makes it easy to keep all of the required components together when making backups of data files.

Creating a FULLTEXT Index

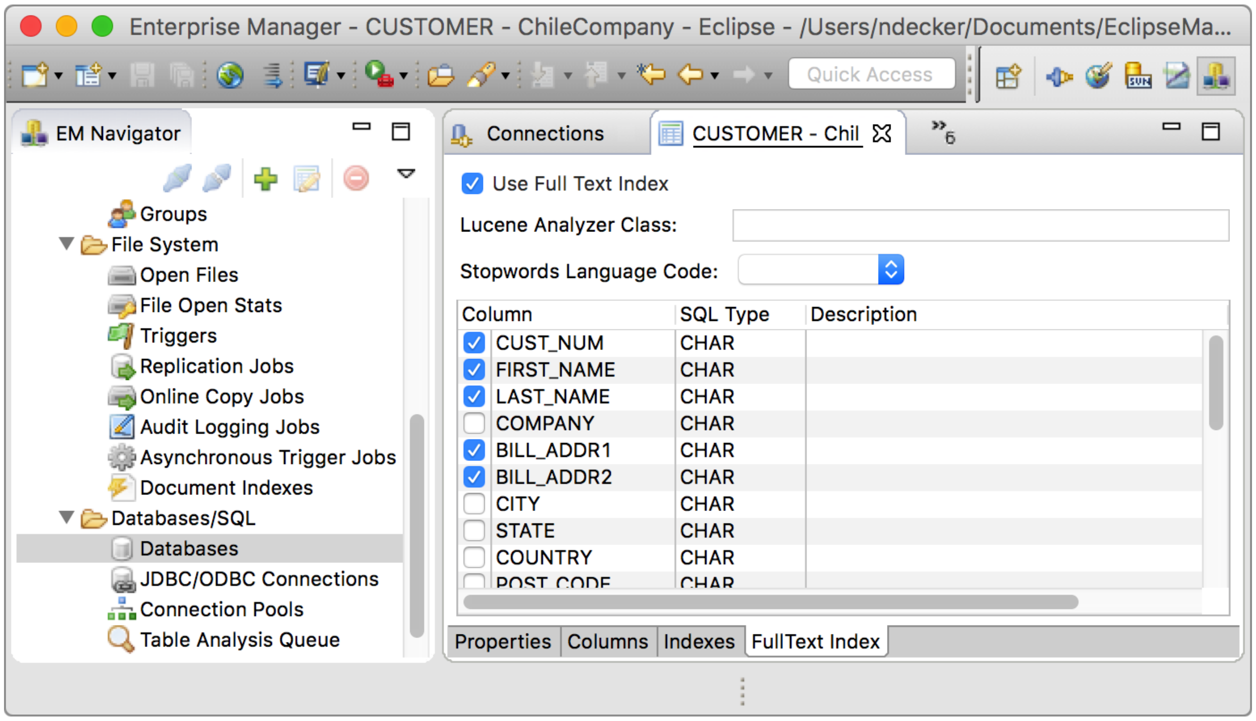
Three options exist to create a FULLTEXT index:

1. SQL's **`CREATE FULLTEXT INDEX`** statement
2. BBj Enterprise Manager (EM)
3. Admin API

This article demonstrates creating a FULLTEXT index using the EM. For information using SQL or the Admin API, see SQL FULLTEXT Indices.

To create a FULLTEXT index on a table using the EM table editor:

1. **Open** the database in the EM.
2. **Select** the Tables tab and double-click on the table to contain the index.
3. **Select** the "FULLTEXT Index" tab.
4. **Click** the "Use FULLTEXT Index" checkbox.
5. **Select** the columns to include in the index, as shown in **Figure 1**.
6. **Save** the changes.



✓ Use Full Text Index

Lucene Analyzer Class:

Stopwords Language Code:

Column	SQL Type	Description
✓ CUST_NUM	CHAR	
✓ FIRST_NAME	CHAR	
✓ LAST_NAME	CHAR	
☐ COMPANY	CHAR	
✓ BILL_ADDR1	CHAR	
✓ BILL_ADDR2	CHAR	
☐ CITY	CHAR	
☐ STATE	CHAR	
☐ COUNTRY	CHAR	
☐ POST_CODE	CHAR	

Figure 1. Creating a FULLTEXT index on the ChileCompany CUSTOMER table

Depending on the number of records in the table, populating the new index with the record data may only take a few moments or a more significant amount of time. Once complete, the index is ready for use and will be dynamically maintained when new records are added, changed or removed.

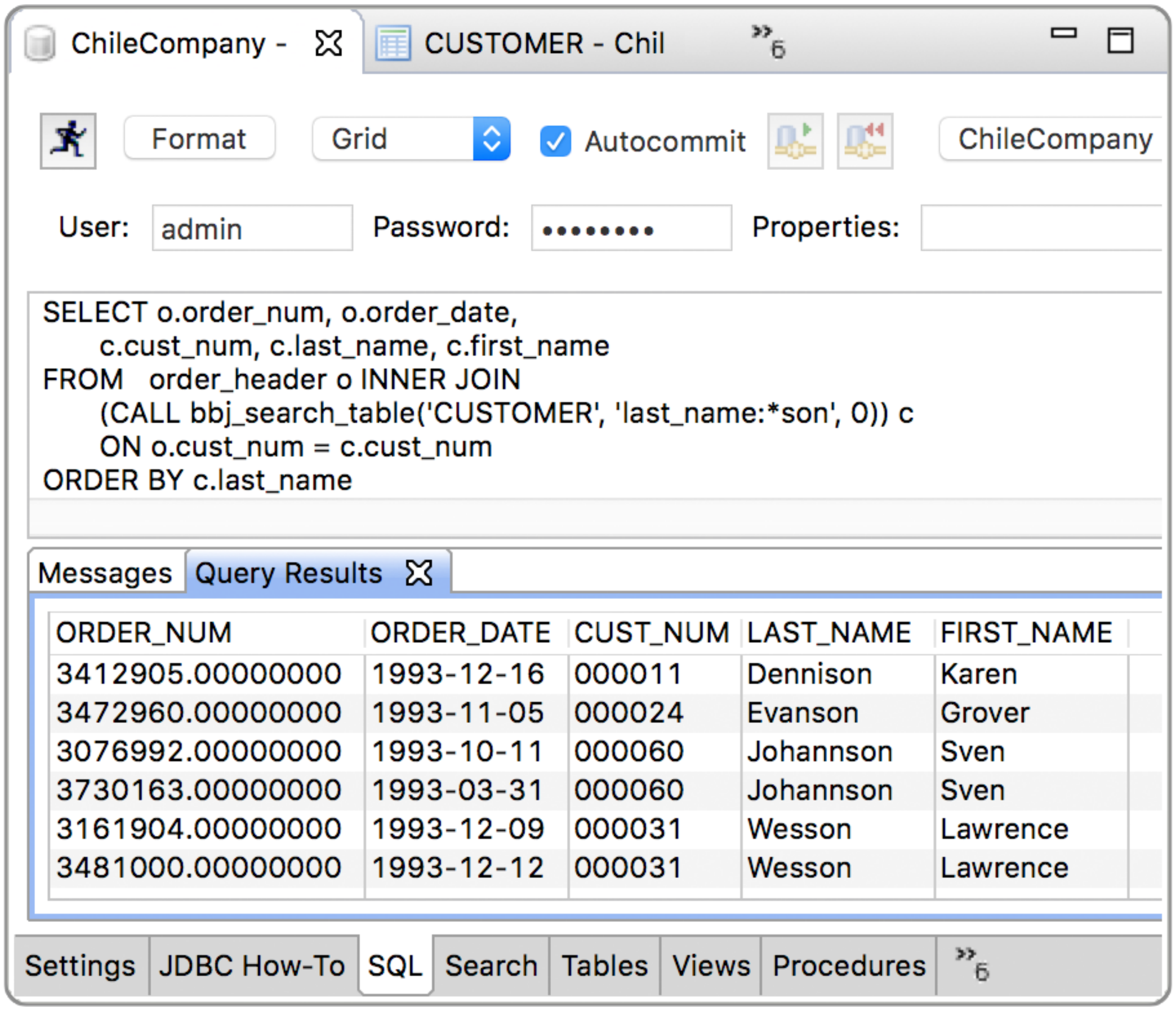
Use a FULLTEXT Index in a Query

With the new FULLTEXT index in place, it is now time to use that index in a query. This example uses the ChileCompany database which comes with FULLTEXT indices in version 15.00 and later. There are four system SPROCs available for use with FULLTEXT indices: **`BBJ_INDEX`**, **`BBJ_SEARCH`**, **`BBJ_SEARCH_TABLE`**, **`BBJ_SEARCH_RELATED`** (covering each in depth is beyond the scope of this article, see Using FULLTEXT Files/Indices For Searching for further information). Let's take a look at the easiest and most common SPROC, **`BBJ_SEARCH_TABLE`**.

`BBJ_SEARCH_TABLE` takes 3 parameters: single table name, Lucene format search query, and maximum number of results. The results from the SPROC are useful standalone, embedded in a SELECT or joined with other tables. This following example finds all customers who have a FIRST_NAME or LAST_NAME that starts with "Ch":

```
SELECT * FROM (CALL bbj_search_table('CUSTOMER', 'Ch*', 0))
```

This first example uses a simple wildcard Lucene query. Lucene provides a robust query language, please see Apache Lucene - Query Parser Syntax for further details and complete information. Another example using a bit more complexity returns all orders with a customer with a LAST_NAME ending with “son” and is shown in **Figure 2.**



User: admin Password: •••••••• Properties:

```
SELECT o.order_num, o.order_date,
     c.cust_num, c.last_name, c.first_name
FROM   order_header o INNER JOIN
     (CALL bbj_search_table('CUSTOMER', 'last_name:*son', 0)) c
     ON o.cust_num = c.cust_num
ORDER BY c.last_name
```

ORDER_NUM	ORDER_DATE	CUST_NUM	LAST_NAME	FIRST_NAME
3412905.00000000	1993-12-16	000011	Dennison	Karen
3472960.00000000	1993-11-05	000024	Evanson	Grover
3076992.00000000	1993-10-11	000060	Johannson	Sven
3730163.00000000	1993-03-31	000060	Johannson	Sven
3161904.00000000	1993-12-09	000031	Wesson	Lawrence
3481000.00000000	1993-12-12	000031	Wesson	Lawrence

Figure 2. Using the FULLTEXT Index to speed up a query with a partial match

Note that in this example the query compares the ending of the last names. With a typical LIKE comparison, the SQL engine will not optimize the comparison and thus examines every record in the table to return the result. On a large table this could take a long time, however, using the FULLTEXT functionality, even a multi-million record table can return the results almost instantly.

Summary

Speed and flexibility are two exceedingly important components of any searching capability in an application. While SQL provides a number of possibilities, the FULLTEXT index functionality now included in BBj rounds out a complete, robust, and lightning fast searching solution for any BBj application.