OmniFind, Part I: Add Sizzle to Your SQL with OmniFind Text Search Server for DB2 for i

Published Tuesday, 10 March 2009 19:00 by MC Press On-line [Reprinted with permission from iTechnology Manager, published by MC Press, LP; http://www.mcpressonline.com.]

Written by Gene Cobb – cobbg@us.ibm.com

With this text search server, you can locate documents that contain specific search strings whether they're in Excel, Word, PDF, PowerPoint, etc.

V6R1 of the IBM i operating system introduced many exciting new features, including numerous DB2 for i enhancements. One such enhancement that has flown somewhat under the radar is the IBM OmniFind Text Search Server for DB2 for i. This new product gives you the power to perform both basic and sophisticated text searching against data that is stored in your DB2 for i database tables. This article introduces this new technology and shows you how to set it up and use it in your environment.

Most companies have important business information stored in a variety of ways. Certainly, a good portion of this information is (or at least should be!) stored in your relational DB2 for i database. However, a wealth of vital data is often stored in other non-relational formats as well. Sales figures and charts in Excel spreadsheets, technical specifications in PDF files, job applicant resumes in Word documents, and strategic initiatives defined in PowerPoint presentations are all good examples of this. Wouldn't it be nice if all of this important, non-relational information could be consolidated in a single relational database? Well, actually it can. Ever since V4R4, you have had the ability to store these types of documents in Large Object Binary (LOB) columns in your database tables.

Hopefully, you're thinking that this notion of storing documents in your DB2 for i tables is a sound one. But in case you aren't yet convinced, consider the following advantages:

- Organization: As mentioned previously, it is very common to have key business information stored in various non-database file formats, such as PDF documents. Rather than have this vital information strewn about various servers, file systems, and/or repositories, you can keep everything organized and centralized in one place.
- Easy retrieval: When these data sources are stored in your database, you can leverage the power of SQL to quickly and easily retrieve the rows the documents are in. For example, use the product number to access both product inventory levels and the product specification PDF document.

- Security: If your documents are scattered throughout your network, you may be exposing sensitive data to those who should not be accessing it. Lock down these documents by using the security features of both the IBM i operating system and DB2 for i. If this is a sensitive part number and access needs to be restricted to a privileged few, you can leverage object-level security and even row-level security (using SQL views) to lock it down.
- A single version of the truth: A common problem in many companies is multiple versions of "the truth." That is, many versions of the same document can exist in various locations throughout your organization. Which version is the master? Because of its ability to centralize and provide easy yet secure access, the database is a logical choice for storing the master copy of your documents. It may not always be possible to enforce a single version of truth, but at the very least, you can establish company policies that require the master version be stored in the database.
- IBM OmniFind for searching: You can now use this text search server to quickly locate documents that contain specific search strings using advanced, linguistic search techniques.

Introducing OmniFind

The OmniFind Text Search Server is a new licensed program product (5733-OMF) that supports rapid text searches on data stored in DB2 text columns or in documents such as PDF files. This provides the same advanced technology previously available on other DB2 family products. OmniFind provides a set of stored procedures to administer the environment, as well as two new integrated SQL functions (SCORE and CONTAINS) that can be used in your SQL statements to specify and execute text-based searches. This all means that OmniFind can be used to find text strings located both in your traditional character fields as well as in documents stored in columns with data types such as LOB. What's more, there's no additional cost for the OmniFind Text Search Server: The product can be ordered at no charge and support for the product is part of your IBM i Software Maintenance agreement.

For example, let's say you have a DB2 for i table called INVENTORY that contains information about the all products you sell. Columns in the table include product_number, product_category, and product_name. In addition, for each product, you also have a PDF file stored in a LOB column named tech_spec_doc. Each PDF file contains all of the technical specification information for that particular product. Now, let's say that you have a requirement to find all products containing the string "headphones" in the PDF document. The "OmniFind-infused" SQL statement would look something like this:

```
SELECT product_number, product_name
FROM inventory
WHERE CONTAINS(tech_spec_doc, 'headphones') = 1
```

With this type of database design and integrated search capability, your DB2 for i database just became even more powerful!

OmniFind Text Indexes

The foundation of the OmniFind technology is the text indexes that are built over the data in the column. These text indexes can be created over a variety of data types that contain plain text, HTML, XML, or many different rich document types. A list of supported data types and rich document types is shown in the table below:

Supported DB2 Data Types and Document Format Types

Supported Column Data Types	Supported Document Format Types
• BINARY • VARBINARY • BLOB • CHAR • VARCHAR • CLOB • DBCLOB • GRAPHIC • VARGRAPHIC	• Plain text • XML • HTML **INSO Document Formats** • Adobe PDF • Rich-Text Format (RTF) • JustSystems Ichitaro • Microsoft Excel • Microsoft PowerPoint • Microsoft Word • Lotus 123 • Lotus Freelance • Lotus WordPro • OpenOffice 1.1 and 2.0 • OpenOffice Calc • Quattro Pro • StarOffice Calc

If you have experience with DB2 for i or any other relational database, you are probably well acquainted with database indexes and their many benefits. The traditional indexes by DB2 for i are binary radix and encoded vector index (EVI). They are used by the database engine for statistical information when formulating an optimal access plan and during the data access implementation. You have probably created many keyed logical files over your physical files and used these indexes in your RPG and COBOL programs. However, the text indexes used by OmniFind are not to be confused with these traditional database indexes. For starters, they are not part of the DB2 for i database. The text indexes actually reside on the text search server within the Integrated File System (IFS). This text search server is created during the product installation process and runs locally in the PASE environment. Once up and running, it communicates with the database via TCP/IP sockets. This environment is shown in Figure 1.

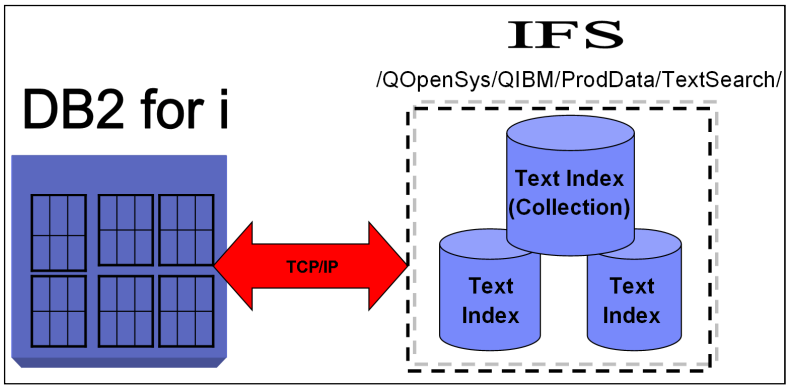

Figure 1: This is the text index architecture.

Another key difference from their DB2 for i counterparts is that the text indexes are not automatically maintained. This is for performance reasons: updating a text-search index can be an extensive process, and keeping it synchronized with table changes automatically could have adverse effects on database performance. Consequently, different approaches to text-index maintenance can be implemented. I'll discuss these later in the article.

Lastly, these indexes cannot be journaled, are not protected by IBM i system-managed access-path protection (SMAPP), and are not backed up using the traditional object-level save commands, such as SAVOBJ and SAVLIB.

Administrative Stored Procedures

The administrative stored procedures are used to enable and disable text searching and to create, update, and drop text indexes. In this section, I will cover each of these stored procedures.

SYSTS_START: Start Text Search Support

Call the SYSTS_START stored procedure to start the text server and enable text search support. The text search server must be enabled for any OmniFind searches to complete successfully.

SYSTS_CREATE: Create a Text Index

The SYSTS_CREATE stored procedure creates a text index for the specified text column, thereby enabling text search indexing for that column. A call to this procedure results in the creation of an object in the IFS text server directory. It also performs the following tasks:

- Creates a view with the same name as the text search index
- Creates a staging table in the QSYS2 schema
- Adds After-Insert, After-Update, and After-Delete triggers to the base table. (I'll explain the roles of these triggers later.)
- Updates the system catalogs with information about the new index

Be aware that calling SYSTS_CREATE does not populate the text index.

An example of calling the SYSTS_CREATE procedure is shown below:

```
CALL SYSPROC.SYSTS_CREATE(
      'myschema',
      'resumes_indx',
      'myschema.resumes(applicant_resume)',
      'FORMAT INSO
       UPDATE FREQUENCY D(*) H(0) M(0)')
```

These are the parameters for this stored procedure:

1. The schema of the text search index
2. The name of the text search index
3. The table and column specification for the document text source (the table schema, table name, and column name)
4. Options

In our example, I specified a couple of values in the Options parameter. The first one is the format. This specifies the content type of the text documents that you intend to index and search. Possible values for this setting are these:

- TEXT
- HTML
- XML
- INSO: This value instructs the OmniFind Text Search Server to determine the format. The format can be any of the supported INSO document formats listed in the table shown at the beginning of this article.

Note: All of the documents in an indexed text column must be of the same format (TEXT, HTML, XML, or INSO). However, if you specify INSO format, the index column can contain multiple document formats (DOC, PDF, XLS, etc.).

Also notice the specification of the "Update Frequency" clause in the above example. This is a purely optional setting that can be used to schedule index updates on the IBM i Job Scheduler. If specified, an entry is placed in the Job Scheduler using the ADDJOBSCDE command. In the above example, the asterisk specified in the Day (D) parameter indicates that the index will be updated every day. If you do not want to schedule your index updates, the alternative would be manual updates by calling the SYSTS_UPDATE procedure.

SYSTS_UPDATE: Update a Text Index

As mentioned, creating the index does not populate it with data. For that, the stored procedure SYSTS_UPDATE must be called. The first time this stored procedure is called for a specific text index, all of the documents (or text strings) from the indexed column are processed and added to the text search index. This initial update requires a full scan of the base table and is depicted in Figure 2.

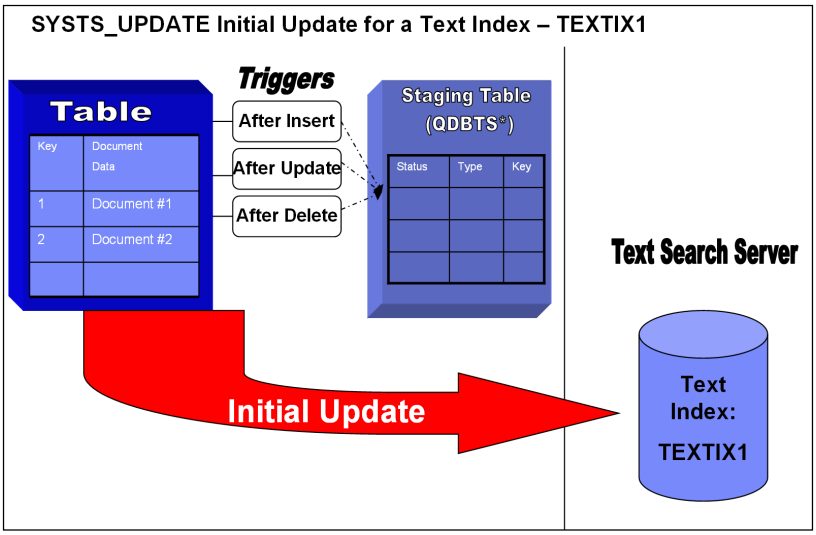

Figure 2: The initial update scans the base table.

While this technique is acceptable for the initial population of the index, a more efficient method is employed to synchronize the index with future document changes to the base table. Recall that when a text index is created, database triggers are added to the base table and a staging table is created. These triggers fire whenever a change occurs over the indexed column in the base table, and they log the information about this incremental update to the staging table.

Because all of these incremental updates are sent to the staging table, subsequent calls to SYSTS_UPDATE result in the processing of the staging table. This is more efficient because it eliminates the need for a full scan of the base table. Instead, only the rows in the staging table are read. The staging table contains the base table key, so each row is joined back to the base table and the text index is updated. This more-resourceful technique is show in Figure 3.

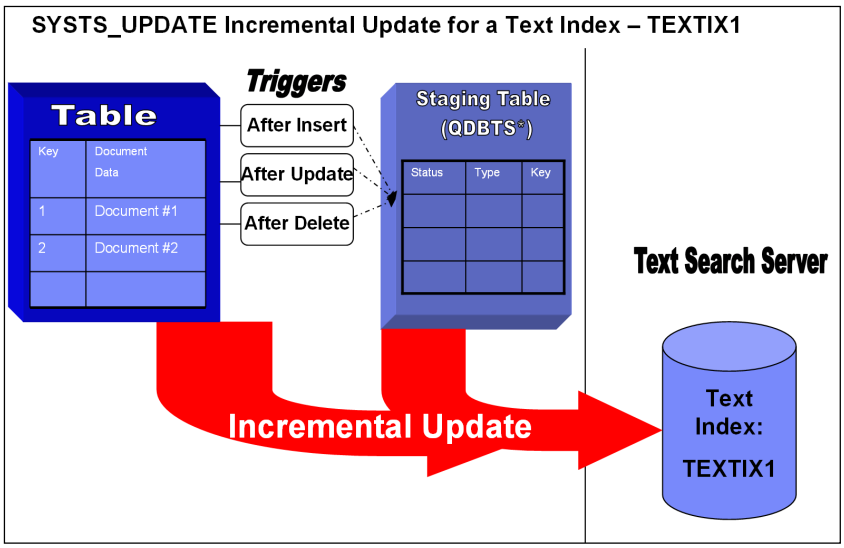

Figure 3: In an incremental update, only the rows in the staging table are read.

If the index was created with the UPDATE FREQUENCY clause, incremental updates will be performed via an IBM i job scheduler entry.

SYSTS_DROP: Drop a Text Index

If you need to drop the text index, the SYSTS_DROP stored procedure will do the trick. This procedure has two parameters: text index name and schema.

If a job scheduler entry was added via the UPDATE FREQUENCY clause, calling this procedure will remove that job scheduler entry for the specified index.

SYSTS_STOP: Stop Text Search Support

As you might have guessed, calling the SYSTS_STOP procedure disables the text search support. While the text search server is down, all SQL requests that include OmniFind built-in functions will fail. However, because database triggers are handling changes to the base tables, all changes continue to be logged to the staging tables.

Searching Using the Built-In Functions

The OmniFind product provides two easy-to-use, integrated built-in functions to help you locate the search strings buried in documents and text fields: CONTAINS and SCORE.

Note: The CONTAINS and SCORE functions are only supported by the SQL Query Engine (SQE).

The CONTAINS function is pretty simple: It accepts (as input parameters) the name of the column, a search argument, and an optional parameter for advanced search options. It searches a text index for the search argument and returns a 1 if a match was found for that row. Otherwise, a 0 is returned.

The following example returns all rows in the INVENTORY_TECH_DOCS table that contain the search argument 'Turntable' in the document stored in the tech_spec_doc column:

```
SELECT productnumber, productname
FROM inventory_tech_docs
WHERE CONTAINS(tech_spec_doc, 'Turntable')=1
```

The SCORE function is similar to CONTAINS, but it actually returns a relevance score that is based on how well a document matches the search argument. A higher score would indicate that more matches were found. The result of SCORE is always a floating decimal value between 0 and 1.

Just like CONTAINS, its input parameters are the name of the column, a search argument, and options. SCORE is often specified as the first ORDER BY column (in descending order) so that the result set shows the top matching rows first. It can also be used in the WHERE clause of a SELECT statement to show only the matches that are higher than a specified minimum value. Because the score is returned as a floating decimal between 0 and 1, you can improve the readability by multiplying the value by 100 and converting it to an integer. An example of this is shown below:

```
SELECT INTEGER(SCORE(tech_spec_doc,'Turntable')*100)
       productnumber, productname
FROM inventory_tech_docs
WHERE SCORE(tech_spec_doc,'Turntable') > .25
ORDER BY SCORE(tech_spec_doc,'Turntable') DESC
```

Once the SQL statement with CONTAINS and/or SCORE is submitted, the search key and options are sent to the Text Search Server. The results are returned to the invoking SQL function. This process is shown in Figure 4.

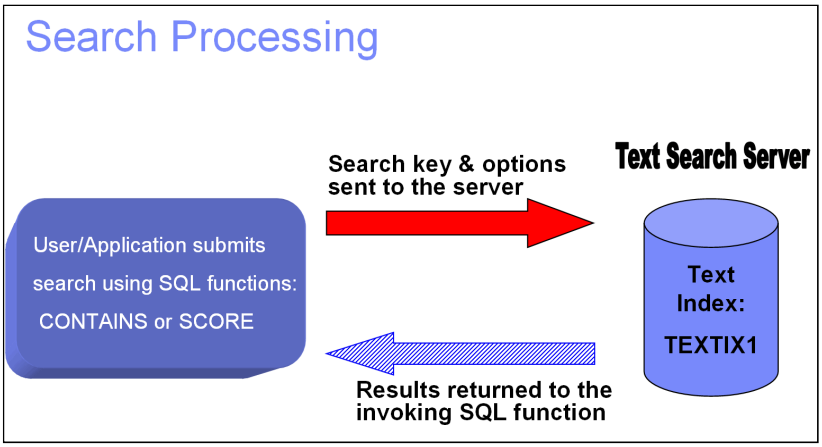

Figure 4: This is how OmniFind search processing works.

Advanced Searching

The above examples are ways you can perform simple searching. In addition, OmniFind allows you to perform more-sophisticated types of text searching:

- AND
- OR
- NOT
- Exact Match
- Wildcard
- Score Boosting
- Includes
- Excludes
- Escape Characters

These operators allow you to extend your searching capabilities and provide your users with ways to find exactly what they are looking for. Some complex query examples are shown in the table below:

Advanced Searching Examples

Operators	Examples	Query Results
" " (Exact Match)	"Java Programming Language"	Returns documents that contain the exact phrase "Java Programming Language."
AND	SQL AND "Java Programming Language"	Returns documents that contain both the term SQL and the exact phrase "Java Programming Language." The AND operator is the default conjunction operator. If no logical operator is between the two terms, the AND operator is used.
OR	"Java Programming Language" OR Java	Returns documents that contain either the exact phase "Java Programming Language" or just Java.
NOT	SQL NOT Java	Returns documents that contain SQL but not Java.
( )	(SQL OR Java) AND XML	Returns documents that contain XML and either SQL or Java . The parentheses ensure that XML is found and either SQL or Java is present.

An Example Implementation

To best illustrate how to set up an OmniFind environment, let's go through an example.

We will take the following steps to configure an OmniFind text search example:

1. Add a LOB column to a table
2. Load PDF documents into the LOB column
3. Start the OmniFind text search server
4. Create and update the text index
5. Use OmniFind functions in SQL statements

Step 1: Add a LOB Column to a Table

As mentioned previously, LOB column support was added in V4R4. However, support for these data types is restricted to SQL only. This means that you cannot create a LOB column using the Data Description Specifications (DDS) language. It also means that you cannot read a table with a LOB column using native record-level access (RLA) in high-level language (HLL) programs such as RPG and COBOL.

SQL is IBM's strategic database language and, as such, is the only interface on the IBM i that provides support for creating and accessing columns with this data type.

To add a LOB column, you have two choices: you could alter an existing table by adding a new column, or you could create a brand new table. See the following examples on how to add a new LOB column.

Note: OmniFind requires that the table have a primary key, unique key constraint, or ROWID column.

New INVENTORY table:

```
CREATE TABLE inventory_tech_docs (
prod_num CHAR(4) NOT NULL DEFAULT '' ,
TECH_SPEC_DOC BLOB(524288000) DEFAULT NULL,
CONSTRAINT INVENTORY_PRODUCTNUMBER
  PRIMARY KEY( prod_num ) )
```

Existing INVENTORY table:

```
ALTER TABLE inventory
ADD COLUMN tech_spec_doc BLOB (524288000)
DEFAULT NULL
```

If you choose the ALTER TABLE method to change the structure of an existing table, the following should be considered:

- Existing programs that use native record-level access (RLA) to read the physical file will experience runtime errors. Because SQL is the only interface that supports LOB columns, any programs with RLA access will not be able to read the file. The only workaround for this is to create logical files or SQL views that exclude the LOB column over the physical file. You would then need to change your programs to read the logical file/view instead.
- Altering a table this way will generate a new format ID (FID) and could necessitate recompilation of programs that use native RLA.

To minimize the potential impact on existing applications, it is recommended that you create a new table with the LOB column. This table would simply be an extension to your existing base table. Join logic could be used to logically merge the two tables together.

Our example uses this type of implementation. From an SQL interface such as the System i Navigator Run SQL script window, enter the following SQL statement:

```
CREATE TABLE inventory_tech_docs (
productnumber CHAR(4) NOT NULL DEFAULT '' ,
TECH_SPEC_DOC BLOB(524288000) DEFAULT NULL,
CONSTRAINT INVENTORY_PRODUCTNUMBER
  PRIMARY KEY( productnumber ) )
```

Next, create an SQL view that joins the INVENTORY table to the new INVENTORY_TECH_SPECS table:

```
CREATE VIEW inventory_omf AS (
SELECT a.prod_num, a.prodname FROM inventory a
INNER JOIN inventory_tech_docs b ON a.prod_num = b.prod_num)
```

Because we did not alter the original INVENTORY file, the programs that use RLA to access this file are not impacted. We can simply use the new INVENTORY_OMF view for all OmniFind-enabled SQL statements.

Step 2: Load Documents into LOB Column

Once you have a table with a LOB column, you can load documents into the column. For this exercise, I mapped a network drive (Q) to my IFS and copied my product technical specification PDF documents into the IFS directory /omniFind/. The name of each PDF file is the product number of the corresponding product. For example, the document for product number 1015 is 1015.pdf. These documents can be seen in Figure 5.

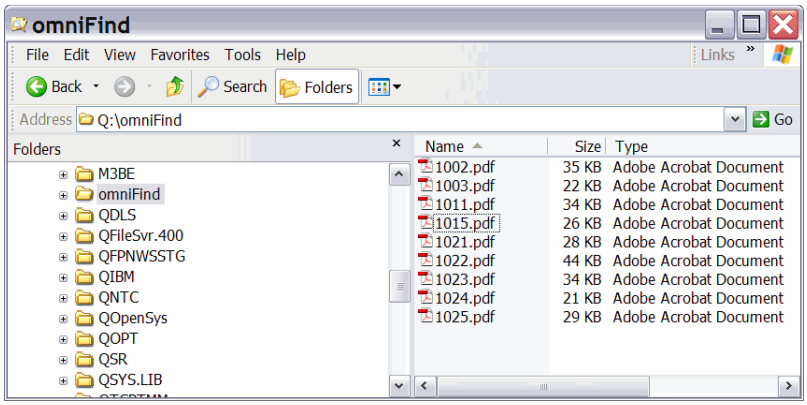

Figure 5: The PDF files are copied to the IFS directory.

Each PDF file is a technical speciation document for a product and is named by the product number. Every product is represented in the INVENTORY table and has a unique key by the PRODUCT_NUMBER column. So what we want to do is load each PDF document into the row for that specific product number. To do this, we need to write a program. This program will need to know what documents are in this IFS directory. Consequently, we need to create a physical file that contains this information.

The following CL program uses the QSHELL command LS to read the contents of the IFS directory /omniFind and populates a physical file named DOCS_LIST (in library QGPL). Each row in DOCS_LIST contains the name of the PDF document in the directory. DOCS_LIST will be read later when loading the documents into the LOB column.

```
PGM
DCL         VAR(&LSCOMMAND) TYPE(*CHAR) LEN(200)
DCL         VAR(&LIBRARY) TYPE(*CHAR) LEN(10)

DLTF        FILE(QGPL/DOCS_LIST)
MONMSG      MSGID(CPF0000)
```

```
CRTPF      FILE(QGPL/DOCS_LIST) RCDLEN(200)
CHGVAR     VAR(&LSCOMMAND) VALUE('LS /omniFind/ > +
           /QSYS.LIB/QGPL.LIB/DOCS_LIST.FILE/DOCS_LIST.MBR')
QSH        CMD(&LSCOMMAND)
ENDPGM
```

Next, issue the following command to compile the program:

```
CRTBNDCL PGM(LISTIFSDIR) SRCFILE(QCLSRC)SRCMBR(LISTIFSDIR)
```

Now, run the program:

```
CALL PGM(LISTIFSDIR)
```

At this point, we have a physical file named DOCS_LIST that contains the names of the documents in the /omniFind directory. The following program can be used to read this file, extract the product number from the document name, and use the product number to find the matching row in the INVENTORY table. If a matching row is found, the PDF document is loaded into the BLOB column and the row is updated. This RPG program (named LOADINV) is shown below:

```
D MYFILE          s                   SQLTYPE(BLOB_FILE)
D selectStm       S            256
d inv_docs_row    ds                  qualified
d  filename
                 200
D SQL_FILE_READ...
D                 c                   const(2)
D SQL_FILE_CREATE...
D                 c                   const(8)
D SQL_FILE_OVERWRITE...
D                 c                   const(16)
D SQL_FILE_APPEND...
D                 c                   const(32)
D work_prod_num   s            200
 /free
    selectStm =
    'SELECT docs_list FROM docs_list';
    EXEC SQL PREPARE S1 FROM :selectStm;
    EXEC SQL DECLARE C1 CURSOR WITH RETURN TO CLIENT FOR S1;
    EXEC SQL OPEN C1;
    dow sqlcod <> 100;
      EXEC SQL
        FETCH c1 INTO :inv_docs_row;
      if sqlcod = 100;
        leave;
      endif;
      // Set the name of the file to load into the column and
      // set the file options setting to SQL_FILE_READ. This
      // tell DB2 to load the file into the column whenever an
      // INSERT or UPDATE is issued
      MYFILE_name = '/omniFind/' + %trim(inv_docs_row.filename);
```

```
        MYFILE_nl   = %len(%trimr(MYFILE_name));
        MYFILE_fo = SQL_FILE_READ;
     // Extract the product number from the name of the document
     // This is used as the product number (unique key)
     // Insert a new row into the inventory_tech_docs table
     work_prod_Num = %subst(%trim(inv_docs_row.filename) : 1 : 4);
     EXEC SQL
       INSERT INTO inventory_tech_docs(prod_num, tech_spec_doc)
            VALUES (:work_prod_num,:MYFILE);
       enddo;
       EXEC SQL CLOSE c1;
       return;
    /end-free
```

After saving this source file member, create the program by issuing the following command from a command line:

```
CRTSQLRPGI OBJ(LOADINV) SRCFILE(QRPGLESRC) SRCMBR(LOADINV)
```

Next, run the program to load the documents into the LOB column:

```
CALL PGM(LOADINV)
```

Now, the PDF documents can be deleted from the IFS directory.

Step 3: Start the Text Search Server

At this point, the TECH_SPEC_DOC column of the INVENTORY_TECH_DOCS table contains the appropriate PDF document, and the OmniFind-specific tasks can begin. First, make sure the text search server is enabled. From an SQL interface, issue the following statement:

```
CALL SYSPROC.SYSTS_START();
```

Step 4: Create and Update the Text Search Index

The next step is to create the text index over the new LOB column. The schema (library) that holds the tables and views is named DATALIB. We will use this schema as the first parameter in the stored procedures to create, update, and drop the text index. The name of our text index will be INV_DOCS_IDX. From an SQL interface, issue the following SQL statement:

```
CALL SYSPROC/SYSTS_CREATE('DATALIB','INV_DOCS_IDX',
'DATALIB.INVENTORY_TECH_DOCS (TECH_SPEC_DOC)',
'CCSID 37   FORMAT INSO')
```

Immediately populate the index by issuing the following SQL statement:

```
CALL SYSPROC/SYSTS_UPDATE('DATALIB','INV_DOCS_IDX', '')
```

If you need to drop the index for any reason, issue the following SQL statement:

```
CALL SYSPROC/SYSTS_DROP('DATALIB ','INV_DOCS_IDX')
```

Step 5: Searching (Use OmniFind Functions in SQL Statements)

At last everything is in place, and the fun part (searching) can begin. Let's say we want to find all technical specification documents in our table that contain the text string

'CD-RW'. We want see the text search score, the product number, and the product name, and we want to sort the list in descending order of the text search score. The following SQL statement would satisfy this requirement:

```
SELECT INTEGER(SCORE(tech_spec_doc,'CD-RW') * 100) AS search_score,
productnumber, productname
FROM inventory_omf
WHERE CONTAINS(tech_spec_doc, 'CD-RW')=1
ORDER BY SCORE(tech_spec_doc,'CD-RW') DESC
```

Notice that we used the SQL view INVENTORY_OMF for this statement. Recall that this view is a join of the INVENTORY table and the INVENTORY_TECH_DOCS table, so it provides an easy interface to access the product name column. The results of this request are shown in Figure 6.

SEARCH_SCORE	PRODUCTNUMBER	PRODUCTNAME
33	1025	Multichannel Super Audio CD Player
23	1021	CD Changer / CD Player
19	1023	400 Disc Super Audio CD Changer

Figure 6: Here are our sample query results.

Additional Information

Obviously, there is much more about OmniFind than could be covered in this article.

A wealth of information can be found in the document "e-business and Web serving OmniFind Text Search Server for DB2 for i5/OS V1.1." It can be downloaded from the IBM i InfoCenter Web site.

But here are several more key points to know about the product:

- It is the strategic replacement for the DB2 for i Text Extender product.
- Its searching algorithms allow for linguistic variations of words. For example, the following SQL statement :

```
SELECT author, story FROM books
WHERE CONTAINS (story, 'mice chasing cats') = 1
returns matches for such variations as "mouse", "mice",
"chase", "chased", "chasing", "cat" and "cats"
```

Another powerful word variation feature is the ability to match on abbreviations and acronyms. For example, a search argument of ‘Iowa’ would return a match even when a document only contains ‘IA’ -

that state's two character abbreviation. Similarly a search argument of "United States" would return matches for documents containing 'USA'.

- It provides XML searching capability. While the other members of the DB2 family support the indexing of XML documents, DB2 for i does not because it does not support the XML data type. However, you can use OmniFind to perform full text searching (targeting specific tags and attributes) on XML documents that are stored in a LOB column of a DB2 for i table.
- Storing the documents in DB2 for i tables is not required. You can set up an environment in which OmniFind can index and search documents stored on the IFS.

OmniFind supports 26 languages (including some double-byte languages, such as traditional Chinese): Arabic, Czech, Danish, German (Switzerland), German (Germany), Greek, English (Australia), English (United Kingdom), US English (United States), Spanish (Spain), Finnish, French (Canada), French (France), Italian, Japanese, Korean, Norwegian Bokmal, Dutch, Norwegian Nynorsk, Polish, Brazilian Portuguese, Portuguese (Portugal), Russian, Swedish, Simplified Chinese, and Traditional Chinese.

Summary

We have covered a lot of ground in this article. You should now be familiar with the benefits of storing your document in LOB columns, understand how to load documents into these columns, and be armed with the knowledge to set up and use the OmniFind product to perform document text searches against these documents. One thing that you may have noticed is that we are missing a graphical interface to perform these searches and actually open one of the matching documents. In Part II of this article, I will show you how to do this by integrating OmniFind with the DB2 Web Query for IBM i product.