

IBM i
7.2

*Electronic business and Web serving
OmniFind Text Search Server for DB2 for i*

IBM

Note

Before using this information and the product it supports, read the information in [“Notices” on page 149.](#)

This document may contain references to Licensed Internal Code. Licensed Internal Code is Machine Code and is licensed to you under the terms of the IBM License Agreement for Machine Code.

© **Copyright International Business Machines Corporation 2002, 2013.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

OmniFind Text Search Server.....	1
What's new for IBM i 7.3.....	1
PDF file.....	1
Introduction.....	2
Overview.....	2
System requirements.....	3
Key concepts.....	4
Create and update index.....	4
Triggers.....	5
Document formats.....	5
Data types.....	6
Score and Synonyms.....	6
Linguistic processing.....	7
Languages.....	7
Chinese, Japanese, and Korean.....	8
Server alias name.....	10
Install and configure.....	11
Install.....	11
Start text search.....	12
Create an index.....	12
Update an index.....	13
Search an index.....	13
Document truncation.....	13
Stored procedures.....	14
SYSTS_START.....	14
SYSTS_STOP.....	15
SYSTS_CREATE.....	17
SYSTS_ALTER.....	26
SYSTS_DROP.....	31
SYSTS_UPDATE.....	32
SYSTS_SHUTDOWN.....	35
Search an index.....	36
CONTAINS.....	36
SCORE.....	39
Search syntax.....	41
Simple examples.....	43
Advanced search operators.....	43
Searching for special characters.....	46
CONTAINS and SCORE example.....	48
XML search.....	49
Namespaces.....	54
Using namespaces.....	55
Example.....	60
Query grammar.....	63
Administer OmniFind.....	64
Start OmniFind.....	64
Stop OmniFind.....	65
Save and restore.....	66
Index structure.....	66
Index with data.....	67
Problem determination.....	69

View logs.....	69
Administration tools.....	69
Configuration tool.....	70
SYSTS_REMOVE.....	74
SYSTS_REPRIMEINDEX.....	76
SYSTS_CLEAR_INDEX.....	77
SYSTS_VALIDITYCHECK.....	78
QDBTS_ROWS_STATUS.....	79
Synonym dictionaries.....	81
Add a synonym dictionary.....	81
Remove a synonym dictionary.....	82
Find orphaned and missing indexes.....	83
Advanced administration.....	84
ServerInstance tool.....	85
Health Checker.....	86
Independent ASP.....	87
High Availability.....	88
Performance analysis.....	89
Transaction considerations.....	92
Using IBM Navigator for i.....	93
Work with servers.....	94
Work with indexes.....	95
View index builds.....	97
Use System i Navigator.....	98
Work with servers.....	99
Work with indexes.....	101
View index builds.....	104
Administration tables.....	105
SYSTEXTDEFAULTS.....	105
SYSTEXTINDEXES.....	106
SYSTEXTCOLUMNS.....	108
SYSTEXTSERVERS.....	109
SYSTEXTCONFIGURATION.....	110
SYSTEXTSERVERHISTORY.....	110
SYSTS_CREATE.....	111
Extensions to Index and Search Non-DB2 Data.....	112
Overview.....	112
Create Collection.....	112
Add Object Set for Spool File Data.....	117
Removing Object Set for Spool File Data.....	121
Add Object Set for a Stream File.....	124
Removing Object Set for Stream File Data.....	126
Adding an Object Set for Multiple Members Source Physical File.....	127
Removing an Object Set for Multiple Members Source Physical File.....	129
Removing an Object Set.....	130
Updating the Collection.....	131
Repopulate the Collection.....	132
Search Collection.....	132
Query Object Sets.....	135
Retrieve Status of Indexes Objects.....	136
Objects Not Indexed.....	137
Retrieve Status of Collection.....	138
Dropping a Text Search Collection.....	139
Altering a Text Search Collection.....	140
IASP Considerations.....	143
Backup and Restore Considerations.....	143
Messages and codes.....	144
OmniFind messages.....	144

Notices	149
Programming interface information.....	150
Trademarks.....	150
 Index	 153

OmniFind Text Search Server for DB2 for i

OmniFind Text Search Server for DB2® for i allows you to issue SQL statements that satisfy familiar text search queries on documents that are stored in a DB2 database.

Note: By using the code examples, you agree to the terms of the [“Code license and disclaimer information”](#) on page 147.

What's new for IBM i 7.3



Read about new or significantly changed information for the OmniFind Text Search Server for DB2 for i topic collection.

The major new features include:

- [Performance tuning parameter extensions to customize configuration settings](#)

How to see what's new or changed

To help you see where technical changes have been made, the information center uses:

- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

To find other information about what's new or changed this release, see the [Memo to users](#).



PDF file for OmniFind Text Search Server for DB2 for i

Use this page to view and print a PDF of this information.

To view or download the PDF file for this document, select [OmniFind Text Search Server for DB2 for i](#) (about 1192 KB).

Other information

You can also view or print any of these PDF files:


- [Preparing for and Tuning the SQL Query Engine on DB2 for i5/OS](#) 
- [SQL Performance Diagnosis on IBM® DB2 Universal Database for iSeries](#) 

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the preceding link).
2. Click the option that saves the PDF locally.
3. Go to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDF files. You can download a free copy from [Adobe](http://get.adobe.com/reader/) (<http://get.adobe.com/reader/>) .

Introduction to the OmniFind Text Search Server for DB2 for i

DB2 for i uses the OmniFind(r) Text Search Server as an indexing and search engine for documents that are stored in a DB2 database.

OMNIFIND is a text search product that allows IBM i users to search unstructured text stored in a column of a DB2 for i table. The text stored in the column can be either simple character text, an XML document, or any of several different types of rich text documents, such as a PDF or DOC file. The product allows users to index unstructured data without having to parse it into a structured form such as an SQL table.

OmniFind Text Search Server is a context-based search engine. It supports fuzzy search capability. For example, a search for 'mice' discovers documents with 'mice' or 'mouse' in them. The search engine also supports language context. For example, it understands the fuzzy search equivalents of 'mice' and 'mouse' in both English and Spanish.

Many applications can take advantage of this capability. A good example is a Human Resources database. Candidate resumes can be stored in the database in whatever form they are submitted. Subsequent searches using OmniFind can be used to search for potential candidates with certain key skills.

Overview of the OmniFind Text Search Server for DB2 for i

OmniFind Text Search Server for DB2 for i provides a set of administrative stored procedures and two built-in functions: CONTAINS and SCORE. These functions are used to search text indexes created from documents stored in a DB2 table. The administrative stored procedures are used to enable and disable text searching and to create, update, and drop text indexes.

A text index can be created over any column of the following data types:

- CHAR
- VARCHAR
- CLOB
- BLOB
- DBCLOB
- GRAPHIC
- VARGRAPHIC
- BINARY
- VARBINARY
- XML

The data can contain plain text, HTML, XML, or many rich document types, such as PDF files. The data is read from the text column and is converted to Unicode (CCSID 1208) before it is indexed.

Text indexes are not typical DB2 indexes. They are not maintained automatically, cannot be journaled, and cannot be backed up using the typical backup and restore methods. Text indexes are created and stored on a text search server.

By default, the text search server is created on the same system as the data stored in the DB2 database. However, a text search server can be created on another server running IBM i, Linux®, UNIX, AIX®, or Windows.

The text search server contains a collection of significant terms extracted from each row of the column. A TCP/IP connection is used to communicate with the text search server.

The CONTAINS and SCORE functions are built-in functions which are integrated into DB2 for i.

DB2 for i uses the OmniFind(r) Text Search Server as an indexing and search engine for documents that are stored in a DB2 database.

OMNIFIND supports multiple collections. A collection contains one text search index and the index-specific options for parsing, indexing, and searching.

OMNIFIND has a graphical user interface for administration of servers and text indexes.

The text search server also provides SQL stored procedures and command-line tools that you can use for common tasks. These common tasks include configuring and administering the text search server, creating a synonym dictionary for a collection, and diagnosing problems.

Related concepts

Administration stored procedures for text search

You can start and stop text search functions and create, drop, and update text search indexes by using a set of administration SQL stored procedures. These procedures can be called from any SQL interfaces. You cannot call these procedures from an IBM i command line by using CL commands.

Related reference

CONTAINS


You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

System requirements for installing the OmniFind Text Search Server for DB2 for i

Before you install an OmniFind Text Search Server for DB2 for i, make sure that your system meets all the hardware, software, and operating system requirements.

When you install OmniFind Text Search Server for DB2 for i, the installation program creates one text search server for IBM i. You can install text search servers on remote servers running Linux or Windows. These servers are part of [DB2 Accessories Suite for z/OS® \(5655-R14\)](http://www-01.ibm.com/software/data/db2imstools/db2tools/accessories-suite/) (<http://www-01.ibm.com/software/data/db2imstools/db2tools/accessories-suite/>) . The link has information about downloading the suite.

Software requirements

Make sure that your system meets the following minimum software requirements:

- 5770JV1 IBM Developer Kit for Java™
 - One of the following:
 - 5770JV1 Option 14 Java SE 7 32 bit
 - 5770JV1 Option 15 Java SE 7 64 bit
- DB2 Universal Java Driver installed and configured on the text search server
- For IBM i, the following programs must be installed:
 - 5770SS1 Option 30 Qshell
 - 5770SS1 Option 33 IBM i Portable Application Solutions Environment (IBM i PASE)
 - 5770SS1 Option 39 International Components for Unicode
 - The latest Group PTF for IBM DB2 for i is applied on the system.

Key concepts

Understanding the key concepts about text search functions helps you to use the benefits of OmniFind Text Search for DB2 for i. Key concepts include the document types and languages that are supported.

Create and update a text search index

You can create a text search index by defining and declaring the properties of the index. You can update a text search index by adding new data from a DB2 table to the index. You can also update a text search index by changing the existing data in the index.

For each text search index that you create, a new collection is created on the OmniFind Text Search Server for DB2 for i. After initial creation, the text search index contains no data.

You add data to the text search index by calling the SYSPROC.SYSTS_UPDATE stored procedure. The first update process adds all the text documents from the text column to the text search index. This process is known as the *initial update*. The subsequent updates are incremental.

When a text search index is created, the following objects are created or updated:

- The staging table is created in the QSYS2 library.
- The INSERT, DELETE, and UPDATE triggers are added to the base table.
- An SQL view with the name of the text search index is created in the schema of the text search index. This view contains information about the text index. For example, the view can be used to obtain the base table name and the staging table name. The view also shows the number of pending changes to the base table that are not yet reflected in the text search index.
- The text search index catalogs (SYSTEXTINDEXES and SYSTEXTCOLUMNS) in the QSYS2 library are updated with a new entry added for the new text search index.

Staging table considerations:

- Do not perform any DB operation on the staging table except saving and restoring the file, or changing authorities.
- If you are changing the authorities on the base table, change the authorities on the staging table also.

Base table considerations:

- Do not remove the DELETE, UPDATE, and INSERT triggers that are added when a text search index is created.
- Dropping the text search index removes the triggers.
- Do not alter or remove the ROWID, primary key, or unique column that was used as the key in the text search index.
- Altering the data column of the base table that results in data truncation might result in false positive matches in the text search index.

Related reference

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

QSYS2.SYSTEXTCOLUMNS administration table

You can see information about the text columns for a text search index in the QSYS2.SYSTEXTCOLUMNS administration table. Each text search index has an index ID, text column names, and the schema name of the base table.

Asynchronous indexing and triggers

You can update the text search index on the OmniFind Text Search Server for DB2 for i manually or schedule it to run automatically.

The text search index maintained on the OmniFind Text Search Server for DB2 for i is not updated synchronously when the DB2 table is updated. Updating a text search index is an extensive operation.

Instead, changes to the DB2 table column are captured by triggers to a local log table. This log table is also called a staging table. These triggers automatically store information about new, changed, and deleted documents in a log table. Each log table is associated with one text search index. Applying the contents of the log table to its corresponding text search index is called an *incremental update*.

You must periodically update the text search index in order for changes to be reflected in queries.

You can update the text search index manually by calling the SYSPROC.SYSTS_UPDATE stored procedure.

Updates can also be scheduled to occur automatically by using the UPDATE FREQUENCY clause on the SYSPROC.SYSTS_CREATE procedure when the text index is created.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

Supported document formats

The text column data can be plain text, an HTML document, an XML document, or any document that is recognized by the search engine.

OmniFind Text Search Server for DB2 for i parses documents to extract the relevant parts and make those parts searchable. For example, tags and metadata in an HTML document are not indexed.

Parsing of the following document formats is supported:

- TEXT: Flat text
- HTML: Hypertext Markup Language
- XML: Extensible Markup Language
- INSO: The OmniFind Text Search Server for DB2 for i uses filters to detect the format of text documents. The following INSO document formats are supported:
 - XML
 - HTML
 - JustSystems Ichitaro
 - Lotus® 123
 - Lotus Freelance
 - Lotus WordPro
 - Microsoft Excel
 - Microsoft PowerPoint
 - Microsoft Rich Text Format

- Microsoft Visio
- Microsoft Word
- Microsoft Write
- Portable Document Format (PDF)
- Quattro Pro
- Rich Text RTF
- StarOffice Calc and OpenOffice Calc

All the documents in an indexed text column must be of the same format (TEXT, HTML, XML, or INSO).

XML data

XML structure in the XML data is indexed in the OmniFind Text Search Server for DB2 for i after parsing the data through an XML parser. Then you can use the supported XML Search query syntax to retrieve the results.

Related concepts

XML search

You can index and search XML documents. The XML search grammar uses a subset of the W3 XPath language with extensions for text search. The extensions support range searches of numeric, Date, and DateTime values that are associated with an XML attribute or element. Structural elements can be used separately, or combined with free text in queries.

Supported data types

The data in the text columns that you want to index and search can be either binary data or character data.

The following data types are binary data:

- BINARY
- VARBINARY
- BLOB

In addition, OMNIFIND handles the following data types similarly to binary data:

- CHAR FOR BIT DATA
- VARCHAR FOR BIT DATA

The following data types are character data:

- CHAR FOR SBCS DATA or FOR MIXED DATA
- VARCHAR FOR SBCS DATA or FOR MIXED DATA
- CLOB
- DBCLOB
- GRAPHIC
- VARGRAPHIC
- XML

If the data is binary data, you can specify the coded character set identifier (CCSID) used to build the text search index. For character data, the DB2 database knows the encoding; therefore, if you explicitly specify a CCSID, that specification is ignored.

Text score and synonym support

You can use synonyms to improve the results for a query. You can use a text score to find out how closely a result matches the query.

Text score

A text score is calculated as part of the search, and can be included in the query results. A text score is a value 0 - 1, up to three decimal points; for example, 0.000 to 1.000. A text score denotes how closely a result matches the query relative to all the other documents in the text search index.

OMNIFIND composes the text score from various factors, such as the general importance of the search terms and the proximity of occurrences of the search terms. The general importance is based on the frequency of the terms in each document and offset by the frequency of the terms across all documents.

Synonym support

The OmniFind Text Search Server for DB2 for i supports the use of synonyms to modify the results of a query. Using synonyms can increase the number of query results by causing more documents to match a query. However, using synonyms might also decrease the precision of a query and make it more difficult to find few documents that match the exact search criteria.

By default, synonyms are not used for a query. To use synonyms for a query, create a synonym dictionary, and add the synonym dictionary to a collection by using the synonym tool.

For more information about synonyms, see [“Synonym dictionaries” on page 81](#).

Related reference

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Linguistic processing

The OmniFind Text Search Server for DB2 for i provides dictionary packs to support the linguistic processing of documents and queries that are not in English.

As an alternative to dictionary-based word segmentation, the OmniFind Text Search Server for DB2 for i uses *n-gram segmentation* support for languages such as Chinese, Japanese, and Korean. *n-gram segmentation* is a method of analysis that considers overlapping sequences of a given number of characters as a single word. Alternatively, Unicode-based *white-space segmentation* uses blank space to delimit words.

If a text document is in one of the supported languages, linguistic processing is carried out when the text is parsed into tokens. For unsupported languages, an error code is returned.

When you search a text search index, a match is indicated that contains linguistic variations of the query terms. The variations of a word depend on the language of the query.

Supported languages

You can specify that text documents be processed using a specific language.

You can specify the language for the indexed text data in the SYSPROC.SYSTS_CREATE administration stored procedure. If you set the value to AUTO, the OmniFind Text Search Server for DB2 for i tries to determine the language. For short documents, automatic detection might be not accurate and is not recommended. The default language for linguistic processing is English (en_US).

The following table shows the five-character language codes for the supported languages.

Language code	Language
ar_AA	Arabic
cs_CZ	Czech
da_DK	Danish

Table 1. The five-character language codes for the supported languages (continued)

Language code	Language
de_CH	German (Switzerland)
de_DE	German (Germany)
el_GR	Greek
en_AU	English (Australia)
en_GB	English (United Kingdom)
en_US	English (United States)
es_ES	Spanish (Spain)
fi_FI	Finnish
fr_CA	French (Canada)
fr_FR	French (France)
it_IT	Italian
ja_JP	Japanese
ko_KR	Korean
nb_NO	Norwegian Bokmal
nl_NL	Dutch
nn_NO	Norwegian Nynorsk
pl_PL	Polish
pt_BR	Brazilian Portuguese
pt_PT	Portuguese (Portugal)
ru_RU	Russian
sv_SE	Swedish
zh_CN	Simplified Chinese
zh_TW	Traditional Chinese

Related reference

`SYSPROC.SYSTS_CREATE`

You can call the `SYSPROC.SYSTS_CREATE` stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the `CONTAINS` or `SCORE` functions.

Linguistic processing for Chinese, Japanese, and Korean documents

You can process documents that are in Chinese, Japanese, or Korean by using dictionary-based segmentation or by using n-gram segmentation.

For a search engine, getting good search results depends in large part on the techniques that are used to process text. After the text is extracted from the document, the first step in text processing is to identify the individual words in the text. Identifying the individual words in the text is referred to as *segmentation*. For many languages, white space (blanks, the end of a line, and certain punctuation) can be used to recognize word boundaries. However, Chinese, Japanese, and Korean do not use white space between characters to separate words, so other techniques must be used.

The OmniFind Text Search Server for DB2 for i provides the following two methods to support the linguistic processing of Chinese, Japanese, and Korean:

- Dictionary-based word segmentation (also called morphological analysis)
- N-gram segmentation

Dictionary-based word segmentation

Dictionary-based word segmentation uses a language-specific dictionary to identify words in the sequence of characters in the document. This technique provides precise search results, because the dictionaries are used to identify word boundaries. However, dictionary-based word segmentation can miss specific matching results.

N-gram segmentation

N-gram segmentation avoids the problem of identifying word boundaries, and instead indexes overlapping pairs of characters. Because the OmniFind Text Search Server for DB2 for i uses two characters, this technique is also called bi-gram segmentation.

N-gram segmentation always returns all matching documents that contain the search terms; however, this technique might sometimes return documents that do not match the query.

By default, the OmniFind Text Search Server for DB2 for i comes with a pre-configured index that uses n-gram segmentation for Chinese, Japanese, and Korean.

To see how both types of linguistic processing work, examine the following text in a document: `election for governor of Kanagawa prefecture`. In Japanese, this text contains eight characters. For this example, the eight characters are represented as A B C D E F G H. A sample query that users might enter could be `election for governor`, which is four characters and are represented as E F G H. (The document text and the sample query share similar characters.)

If you use n-gram segmentation processing:

After the document is indexed, the search engine segments the text `election for governor of Kanagawa prefecture` into the following sets of characters: AB BC CD DE EF FG GH

The sample query `election for governor` is segmented into the following sets of characters: DE EF FG GH. If you search with the sample query `election for governor`, the document is found. The reason is that the tokens for both the document text and the query appear in the same order.

When you enable n-gram segmentation, you might see more results but possibly less precise results. For example, in Japanese, if you search with the query `Kyoto` and a document in your index contains the text `City of Tokyo`, the document is found. The reason is that `City of Tokyo` and `Kyoto` share two of the same Japanese characters.

If you do not use n-gram segmentation processing:

After the document is indexed, the search engine segments the text `election for governor of Kanagawa prefecture` into the following sets of characters: ABC DEF GH.

The sample query `election for governor` is segmented into the following sets of characters: EF GH. The characters EF do not appear in the tokens of the document text. (Even though the document does not have EF, it does have DEF).

The document text contains DEF, but the query contains only EF. Therefore, the document is less likely to be found by using the sample query.

When you do not enable n-gram segmentation, you probably receive more precise results but possibly fewer results.

Server alias name

You can use a server alias name to assign a meaningful name to a server.

Each text search server is uniquely identified by column SERVERID in catalog QSYS2.SYSTEXTSERVERS. The SERVERID column is an incrementally generated integer by database.

The ALIASNAME column in QSYS2.SYSTEXTSERVERS is provided to allow a meaningful alias name to be assigned to each server. Server alias names can be used to refer to servers in SYSTS_START, SYSTS_STOP and SYSTS_CREATE , SYSTS_SHUTDOWN and SYSTS_CLEAR_INDEXES procedures.

Note: Server alias name can be changed directly in the QSYS2.SYSTEXTSERVERS catalog table by updating the ALIASNAME value.

Assign server number 1 an alias name of "PRIMARY_LOCAL_SERVER".

```
UPDATE QSYS2.SYSTEXTSERVERS
SET ALIASNAME = 'PRIMARY_LOCAL_SERVER'
WHERE SERVERID = 1
```

To remove an alias name from a server, set the column to NULL.

Remove an alias name from server number 1.

```
UPDATE QSYS2.SYSTEXTSERVERS
SET ALIASNAME = NULL
WHERE SERVERID = 1
```

Related reference

[QSYS2.SYSTEXTSERVERS administration table](#)

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

[SYSPROC.SYSTS_START](#)

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

[SYSPROC.SYSTS_STOP](#)

You can call the SYSPROC.SYSTS_STOP stored procedure to stop DB2 text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped).

[SYSPROC.SYSTS_CREATE](#)

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

[SYSPROC.SYSTS_SHUTDOWN](#)

You can call the SYSPROC.SYSTS_SHUTDOWN stored procedure to shutdown DB2® text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped) and also ends the text search server jobs on the host system.

[SYSTS_CLEAR_INDEXES](#)

You can remove orphaned indexes with the SYSPROC. SYSTS_CLEAR_INDEXES SQL stored procedure. Another implicit way is invoking SYSTS_START directly, which tries to clear orphaned indexes automatically.

Install and configure text search functions

You can install and configure OmniFind Text Search Server for DB2 for i. You can also create and update a text search index so that you can start using text search functions against a column in a table.

Install OmniFind Text Search Server for DB2 for i

Install OmniFind Text Search Server for DB2 for i using the standard installation procedures for a licensed program on IBM i. A default text server is created, and the QSYS2.SYSTEXTSERVERS table is populated with default server information. Optionally, you can add additional text search servers after the install.

Populate the QSYS2.SYSTEXTSERVERS table

OmniFind Text Search Server for DB2 for i is the licensed program 5733-OMF from IBM. See [Installing additional licensed programs](#) for details about how to install a licensed program. To find this product, enter **GO LICPGM** from the command line and select option 10 (Display installed licensed programs). It is displayed under the list of licensed programs.

The QSYS2.SYSTEXTSERVERS table contains information about the IBM OmniFind Text Search Servers installed and available for DB2 for i. When the OmniFind Text Search Server for DB2 for i product is first installed, a default text server is created on the IBM i system. The QSYS.SYSTEXTSERVERS table is also populated with default server information.

Create additional text search servers

If you are using text search servers on a remote IBM system, or if you are using non-IBM servers such as a Windows or Linux server, explicitly populate this table by issuing an SQL INSERT statement.

If you want to populate the QSYS2.SYSTEXTSERVERS table with additional servers, follow these steps on the SQL INSERT:

1. Specify the server port number and server name for each text search server on the SERVERPORT column and SERVERNAME columns of the QSYS2.SYSTEXTSERVERS table by issuing an SQL INSERT statement.
2. Specify the authentication token from each text search server on the SERVERAUTHTOKEN column of the QSYS2.SYSTEXTSERVERS table on the SQL INSERT statement.

When the DB2 database communicates with a text search server, an authentication token is required. This token is generated on the text search server during the installation.

3. Specify the server key for each text search server on the SERVERMASTERKEY column of QSYS2.SYSTEXTSERVERS table SQL INSERT statement.
4. OPTIONAL: Specify an ALIASNAME for the server on the ALIASNAME column. This ALIASNAME can be used to refer to the server on subsequent operations.

Example

The following example of an SQL INSERT statement copies the required information for a text search server to the columns in the QSYS2.SYSTEXTSERVERS table:

```
INSERT INTO QSYS2.SYSTEXTSERVERS (SERVERNAME,
                                   SERVERADRINFO,
                                   SERVERPORT,
                                   SERVERTYPE,
                                   SERVERAUTHTOKEN,
                                   SERVERMASTERKEY,
                                   SERVERPATH,
                                   ALIASNAME)
VALUES ('127.0.0.1',
```

```
VARBINARY(X'0000'),
49200,
0,
'AH2X4w==',
'b1YhcR90858A1wxLJeIY/Q==',
'/QOpenSys/QIBM/ProdData/TextSearch/server1/bin/',
'LOCALSERVER2');
```

The example values must be replaced with the actual values for the server.

Related concepts

Server alias name

You can use a server alias name to assign a meaningful name to a server.

Related reference

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

Start text search functions

Before you start using the text search functions, call the SYSPROC.SYSTS_START stored procedure. By calling this procedure, you can start all the production servers that you have defined to be local to the system.

Text search support includes SQL statements that use the CONTAINS function, the SCORE function, and the following administration stored procedures:

- SYSPROC.SYSTS_CREATE
- SYSPROC.SYSTS_UPDATE
- SYSPROC.SYSTS_DROP

Related reference

SYSPROC.SYSTS_START

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

SYSPROC.SYSTS_DROP

You can call the SYSPROC.SYSTS_DROP stored procedure to drop a text search index that was defined by using the SYSPROC.SYSTS_CREATE stored procedure.

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Create a text search index

You can create a text search index by calling the SYSPROC.SYSTS_CREATE stored procedure.

The DB2 base table must contain a ROWID column, unique key, or primary key.

To create a text search index on an existing DB2 table with a column that contains text:

Call the SYSPROC.SYSTS_CREATE stored procedure.

The text search index is empty until the first time that you update the index.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

Update a text search index

You can update a text search index by calling the SYSPROC.SYSTS_UPDATE stored procedure.

The SYSTS_UPDATE call is used to initially populate the text search index. It is also used any time the contents of the DB2 table changes and you want to synchronize the text search index to those changes.

After a text search index update occurs, you can perform search queries on the text search index. The base table text search column can be changed after the update. In this case, the search query results do not reflect those changes until the next text search index update is run.

Related reference

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

Search a text search index

You can search a text search index by using an SQL statement with a CONTAINS or SCORE function. The search argument criteria is specified on the function.

The user who is performing the text queries on a DB2 table must have the standard privilege set that is required for any form of query, as specified in the *DB2 SQL Reference*.

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Document truncation

The OmniFind Text Search Server for DB2 for i limits the number of characters that can be indexed for each text document. Sometimes this character limit results in the truncation of large text documents in the text search index.

Documents that contain more than 10 million Unicode characters might be truncated by the text search server. For a rich text document, this limit is applied after the document is transformed to plain text.

If a text document is truncated during the parsing stage, you receive a warning that some documents were not processed completely. The warning appears in the job log. The document is partially indexed. Text that is in the document after the limit is reached is not indexed and is not considered during searches.

You might want to remove the document that has been truncated from the text search index to avoid unexpected behavior during search processing. You can remove the document by removing the corresponding record from the DB2 table, or by changing the value for the document to empty or null.

Administration stored procedures for text search

You can start and stop text search functions and create, drop, and update text search indexes by using a set of administration SQL stored procedures. These procedures can be called from any SQL interfaces. You cannot call these procedures from an IBM i command line by using CL commands.

When looking at the system catalogs using STRSQL, you can see the content of the columns by setting your job to a CCSID other than 65535.

SYSPROC.SYSTS_START

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

Text search functions include support for SQL queries that use the CONTAINS function, the SCORE function, and the administration stored procedures that are used to maintain text search indexes.

Run the SYSPROC.SYSTS_START stored procedure each time when a server is added or changed in QSYS2.SYSTEXTSERVERS table.

If text search functions are not started, the database returns SQLCODE -20424 with reason code 4 for the CONTAINS and SCORE functions. The SYSPROC.SYSTS_CREATE and SYSPROC.SYSTS_UPDATE administrative procedures also fail with SQLCODE -20424 if the server has not been started.

For the text search servers that are contained in the QSYS2.SYSTEXTSERVERS table, TCP/IP names are resolved. Multiple calls to the SYSPROC.SYSTS_START stored procedure are not considered an error. This process allows you to verify the address resolution in the QSYS2.SYSTEXTSERVERS table.

If the server is a local server, then this stored procedure call starts the server if it is not already started. If the server is a remote server, the procedure call verifies that the server is active, but does not actually start the server.

Prerequisites

Before you call the SYSPROC.SYSTS_START stored procedure, verify that the QSYS2.SYSTEXTSERVERS table contains at least one entry.

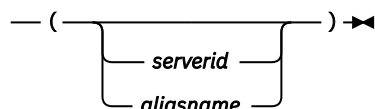
Authorization

The user ID under which this stored procedure is invoked must have the following privileges:

- *EXECUTE authority on the procedure
- SELECT and UPDATE privilege on the SYSTXTSRVR table.
- *EXECUTE authority on the QSYS2 library of the SYSTXTSRVR file.
- *JOBCTL authority or QIBM_DB_SQLADM security special function usage.

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

► SYSPROC.SYSTS_START () ►

The schema qualifier is SYSPROC.

Parameter

serverid or *aliasname*

Specifies the identifier of the server to be started. A *serverid* or server *aliasname* is a string. If no identifier is provided, the default is to start all servers. The identifier string must either be a valid *serverid* that exists in the SERVERID column, or a valid server *aliasname* that exists in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS table. If the identifier can be converted to an integer value, it is interpreted as a *serverid*. If the identifier cannot be converted to an integer value, it is interpreted as a server *aliasname*.

The data type of this parameter is VARCHAR(128).

Note: Only the servers that are identified as production servers are started if no value is specified for *serverid* or *aliasname*. Production servers are identified by the parameter **SERVERCLASS** = 0 in the QSYS2.SYSTEXTSERVERS table. Any test servers must be started by specifying the *serverid* or *aliasname* that is associated with the test server.

To start all production servers:

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

To start a server with an ID of 1:

```
CALL SYSPROC.SYSTS_START(1)
```

To start a server with an alias name of "LOCAL_SERVER":

```
CALL SYSPROC.SYSTS_START('LOCAL_SERVER')
```

To start a server with an alias name of "local_server":

```
CALL SYSPROC.SYSTS_START('local_server')
```

Related concepts

[Server alias name](#)

You can use a server alias name to assign a meaningful name to a server.

Related tasks

[Start the OmniFind Text Search Server for DB2 for i](#)

You can start the OmniFind Text Search Server for DB2 for i by calling SYSPROC.SYSTS_START.

Related reference

[QSYS2.SYSTEXTSERVERS administration table](#)

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

[CONTAINS](#)

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

[SCORE](#)

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

SYSPROC.SYSTS_STOP

You can call the SYSPROC.SYSTS_STOP stored procedure to stop DB2 text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped).

After this stored procedure has completed, SQL queries that use the CONTAINS or SCORE functions or administration stored procedures used for index maintenance return a failure without trying to contact a text search server.

Note: Administrative procedures and SQL queries using the CONTAINS or SCORE built-in functions that were running when SYSTS_STOP was invoked are allowed to complete.

Changes to the based-on table of the index continue to be logged, even when the server is stopped. However, scheduled updates of the index do not occur until SYSPROC.SYSTS_START has been invoked.

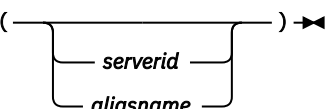
Authorization

The user ID under which this stored procedure is invoked must have the following privileges:

- *EXECUTE authority on the procedure
- SELECT and UPDATE privileges on the SYSTEXTSERVERS table.
- *EXECUTE authority on the QSYS2 library of the SYSTEXTSERVERS file.
- *JOBCTL authority or QIBM_DB_SQLADM security special function usage.

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

► SYSTEXTSERVERS.STOPL — () ►

The schema qualifier is SYSPROC.

Parameter

serverid or *aliasname*

Specifies the identifier of the server to be stopped. A *serverid* or server *aliasname* is a string. If no identifier is provided, the default is to stop all servers. The identifier string must either be a valid *serverid* that exists in the SERVERID column, or a valid server *aliasname* that exists in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS table. If the identifier can be converted to an integer value, it is interpreted as a *serverid*. If the identifier cannot be converted to an integer value, it is interpreted as a server *aliasname*.

The data type of this parameter is VARCHAR(128).

To stop all production servers:

```
CALL SYSPROC.SYSTEXTSERVERS.STOPL()
```

To stop a server with an ID of 1:

```
CALL SYSPROC.SYSTEXTSERVERS.STOPL(1)
```

To stop a server with an alias name of "LOCAL_SERVER":

```
CALL SYSPROC.SYSTEXTSERVERS.STOPL('LOCAL_SERVER')
```

To stop a server with an alias name of "local_server":

```
CALL SYSPROC.SYSTEXTSERVERS.STOPL('local_server')
```

Related concepts

[Server alias name](#)

You can use a server alias name to assign a meaningful name to a server.

Related reference

[QSYS2.SYSTEXTSERVERS administration table](#)

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

[SYSPROC.SYSTEXTSERVERS.START](#)

You can enable DB2 text search functions by calling the SYSPROC.SYSTEXTSERVERS.START stored procedure.

[CONTAINS](#)

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

[SCORE](#)

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

The text search index is created on one of the text search servers that is listed in the QSYS2.SYSTEXTSERVERS table. The text search index is not updated synchronously when the DB2 table is updated. Instead, a log of changes to the DB2 table column is captured by triggers and placed in a staging table.

Note: This stored procedure only defines the text search index. The text search index does not contain any data until after the first invocation of the SYSPROC.SYSTS_UPDATE stored procedure for the new text search index. You create the text search index after the table is initially populated. By creating the text search index after the table is initially populated, you avoid the firing of change triggers before an initial index update.

Prerequisites

Before the SYSPROC.SYSTS_CREATE stored procedure call, verify the following prerequisites:

- DB2 text search functions were started by invoking the SYSPROC.SYSTS_START stored procedure and at least one text search server is running.
- The table includes a column that is defined as primary key, unique index, or ROWID.
- The QSYS2.SYSTEXTSERVERS table contains at least one entry.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The privilege to create in the schema. For more information, see [Authorization, privileges and object ownership](#).
- Administrative authority

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The following system authorities:
 - *USE to the Create Logical File (CRTLF) command
 - *CHANGE to the data dictionary if the library into which the text index is created is an SQL schema with a data dictionary
- Administrative authority

The privileges held by the authorization ID of the statement must also include at least one of the following privileges:

- For the referenced table:
 - The INDEX privilege on the table
 - The system authority *EXECUTE on the library containing the table
- Administrative authority
- If SQL names are specified, and a user profile exists that has the same name as the library into which the text index is created, and that name is different from the authorization ID of the statement, then the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The system authority *ADD to the user profile with that name
- Administrative authority

If a distinct type is referenced, the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- For each distinct type identified in the statement:
 - The USAGE privilege on the distinct type, and
 - The system authority *EXECUTE on the library containing the distinct type
- Administrative authority

If the column that the text search index is built over contains a field procedure, the authorization ID must have one of

- *EXECUTE authority to the program and
- *EXECUTE authority on the library containing the program

Or

- Administrative authority

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

```

SYSYS_CREATE ( ( indexSchema , indexName , textSource ,
               null
             )
  options )

```

The schema qualifier is SYSPROC.

Parameters

indexSchema

Identifies the schema of the text search index. If this parameter is null, the value of the CURRENT SCHEMA special register for the invoker is used. This value must be a valid SQL name.

Note: Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords that can be used.

The data type of this parameter is VARCHAR(128).

indexName

Identifies the name of the text search index. The name of the text search index with the index schema uniquely identifies the text search index in the DB2 subsystem. You must specify a non-null value for this parameter. This value must be a valid SQL name.

Note: Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords that can be used.

The data type for this parameter is VARCHAR(128).

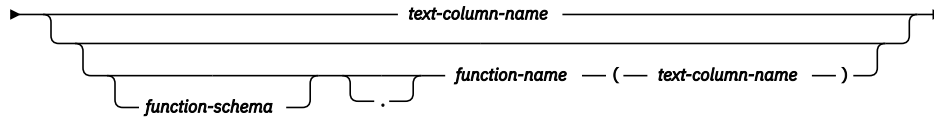
textSource

Identifies the table and column specification for the document text source. This parameter can include user-defined functions. You must specify a non-null value for this parameter.

The data type for this parameter is VARCHAR(1024).

textSource

► *tableSchema* . *tableName* — (—



►) ►

tableSchema

Identifies the schema of the table on which the text search index is created.

Note: Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords that can be used.

tableName

Identifies the name of the text table that contains the column that the external text search index is created on.

Notes:

- Views and logical files are not supported.
- An alias must point to a table or a single member of a physical file.
- Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords.

text-column-name

Identifies the name of the column that contains the text that is used for creating the text search index. This column must be of type CHAR, CHAR FOR BIT DATA, BINARY, VARCHAR, VARCHAR FOR BIT DATA, VARBINARY, CLOB, DBCLOB, BLOB, XML, GRAPHIC, or VARGRAPHIC. If the data type is not one of these data types, you can specify an external function that returns a supported data type.

Notes:

- Only one text search index is allowed for a column. If a text search index exists for the column, SQLCODE-20427 is returned.
- Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords.

function-schema. function-name

Identifies the schema and the name of a built-in or user-defined function. The function can be used to modify a text document stored in the column. The function can also be used to access text documents in a column that is not of a supported data type. Or the function can be used to access a document that is stored elsewhere. The function has one input parameter for the text column data type. For example, an integer that serves as a foreign key to the document content in another table. The function returns a value of one of the OmniFind Text Search for DB2 for i supported data types. The function transforms the text column content to the indexed document content.

Notes:

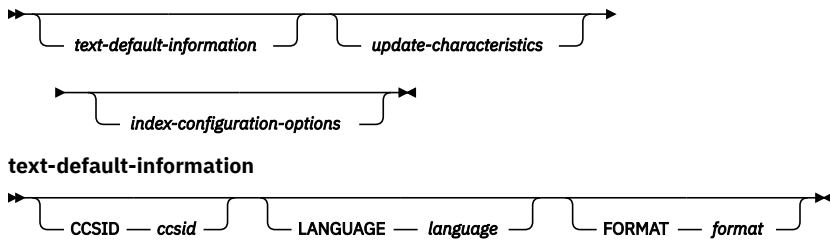
- Cast functions and functions with more than one argument are not allowed.
- Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords that can be used.

options

A character string that specifies the various options that are available for this stored procedure.

The data type for this parameter is VARCHAR(32000).

options



text-default-information

Specifies the coded character set identifier used when indexing binary text documents. Also specifies the language that is used when processing documents, and the format of text documents in the column.

CCSID *ccsid*

Specifies the coded character set identifier that is used for a text search index in a column with a binary data type. The default value is 1208 (UTF-8) and is taken from the QSYS2.SYSTEXTDEFAULTS table. All the CCSIDs that are supported for conversion to UTF-8 by IBM i conversion services are allowed for this parameter.

This parameter is ignored for a text search index in a column with a non-binary data type. Text columns inherit the CCSID from the table specification. The *ccsid* value is ignored when the *format* value is set to INSO.

LANGUAGE *language*

Specifies the language that OmniFind Text Search Server for DB2 for i uses for the linguistic processing of text documents. The default value is en_US (English). If you specify the value for this parameter as AUTO, OmniFind Text Search Server for DB2 for i tries to determine the language.

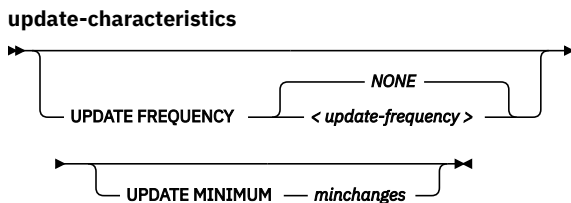
Important: If the language of the documents is not English, do not use the default value of en_US. Change the value to the language of the documents; otherwise, linguistic processing does not work as expected.

FORMAT *format*

Identifies the format of text documents in the column, such as HTML. The OmniFind Text Search Server for DB2 for i needs to know the format, or content type, of the text documents that you intend to index and search. If you do not specify the *format* parameter, the default value is taken from the FORMAT column in the QSYS2.SYSTEXTDEFAULTS table. The supported *format* values are TEXT, HTML, XML, and INSO.

The *format* value INSO allows OmniFind Text Search Server for DB2 for i to determine the format. In this case, the *ccsid* value is ignored. If the OmniFind Text Search Server for DB2 for i cannot determine the document format, then a document error is noted in the job log during processing by the SYSPROC.SYSTS_UPDATE stored procedure.

Note: If you do not specify the *format* parameter while creating the index over an XML data type column, the default value is XML. If you specify the *format* parameter as TEXT or INSO, the XML search capability is not available over this index. In addition, a warning message appears in the job log.



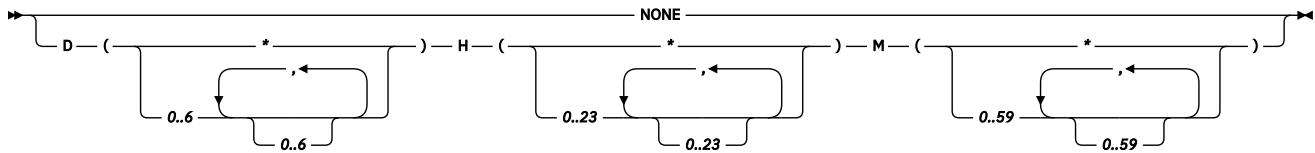
update-characteristics

Specifies the frequency of automatic updates to the text search index. Also specifies the minimum number of changes to text documents before the text search index is updated incrementally at the specified time.

UPDATE FREQUENCY *update-frequency*

Specifies when to make automatic updates to the text search index. The default value is NONE. This option might be useful for a text column in which there are no further changes. The format of the update-frequency option supports two different formats.

update-frequency (Format 1)



NONE

If NONE is specified, then no further index updates are made. The update must be started manually. This option might be useful for a text column in which no further changes are planned.

D

Specifies the day or days of the week when the index is updated. An asterisk (*) specifies all days. 0 specifies Sunday.

H

Specifies the hour or hours when the index is updated. An asterisk (*) specifies all hours.

M

Specifies the minute or minutes when the index is updated. An asterisk (*) cannot be specified. The minimum update frequency is 5 minutes.

Example: This example specifies that the index update is to run every 30 minutes.

```
UPDATE FREQUENCY D(*) H(*) M(0,30)
```

update-frequency (Format 2, chronological)

► <minute> — <hour> — <dayOfMonth> — <monthOfYear> — <dayOfWeek> ►

The format of the *update-frequency (chronological)* option is a list of the five values separated by a blank space. The five values represent the minutes, hours, days of the month, months of the year, and days of the week beginning with Sunday.

If you specify an interval of values or an asterisk (*), you can specify a step value by using a forward slash (/) at the end of the defined interval.

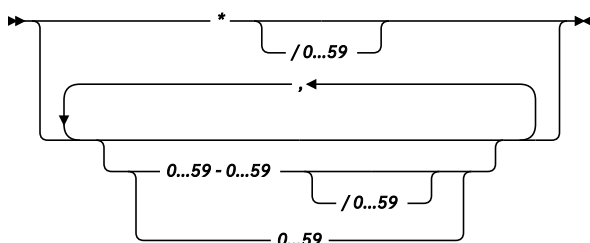
Example: This example specifies that the index update is to run every quarter hour (0, 15, 30, 45) on the even hours between 8 a.m. and 6:45 p.m. (8-18/2 is equivalent to 8, 10, 12, 14, 16, 18), from Monday to Friday every month of the year (* * 1-5).

```
0,15,30,45 8-18/2 * * 1-5
```

minute

Specifies the minutes of the hour when the text search index is to be updated. You can specify an asterisk (*) for an interval of every 5 minutes, or you can specify an integer 0 - 59. You cannot repeat values. The minimum update frequency is 5 minutes. A value of 1,4,8 is not valid.

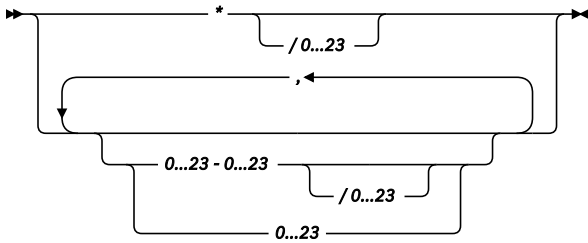
update-frequency (minute)



hour

Specifies the hours of the day when the text search index is to be updated. You can specify an asterisk (*) for every hour, or you can specify an integer 0 - 23. You cannot repeat values.

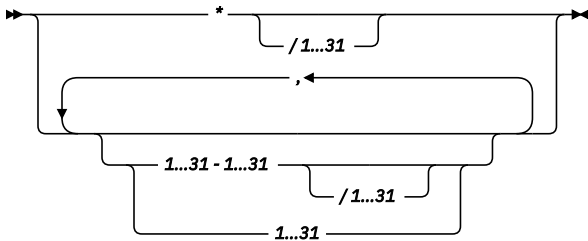
update-frequency (hour)



dayOfMonth

Specifies the days of the month when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 1 - 31. You cannot repeat values.

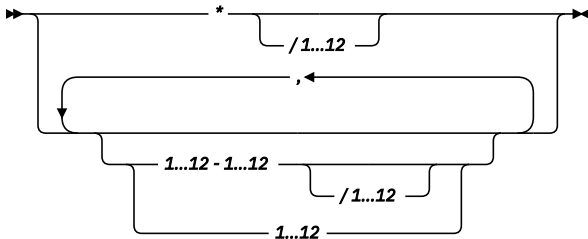
update-frequency (dayOfMonth)



monthOfYear

Specifies the months of the year when the text search index is to be updated. You can specify an asterisk (*) for every month, or you can specify an integer 1 - 12. You cannot repeat values.

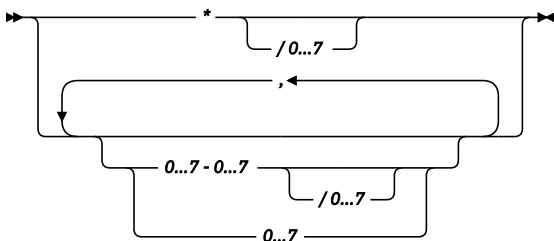
update-frequency (monthOfYear)



dayOfWeek

Specifies the days of the week when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 0 - 7. Both 0 and 7 are valid values for Sunday. You cannot repeat values.

update-frequency (dayOfWeek)



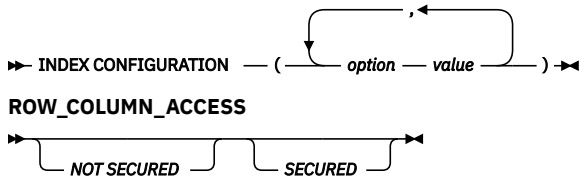
UPDATE MINIMUM minchanges

Specifies the minimum number of record changes made to the underlying table before the text search index is updated incrementally at the time specified in the *update-frequency*

option. The value must be an integer 1 - 2147483647. The default value is taken from the UPDATEMINIMUM column in the QSYS2.SYSTEXTDEFAULTS table.

This option is ignored when you update the text search index, unless you specify the USING UPDATE MINIMUM option in the SYSPROC.SYSTS_UPDATE stored procedure.

index-configuration-options



index-configuration-options

Specifies additional index-specific values as option value pairs. You must enclose string values in single quotation marks. A single quotation mark within a string value must be represented by two consecutive single quotation marks.

CJKSEGMENTATION

Specifies the segmentation method to use when you index documents for CJK (Chinese, Japanese, Korean) languages. The supported values are MORPHOLOGICAL and NGRAM. If the CJKSEGMENTATION value is not specified, the default value is used. The default value is specified by the DEFAULTNAME value in the QSYS2.SYSTEXTDEFAULTS table.

COMMENT

Specifies a comment that is stored in the REMARKS column of the QSYS2.SYSTEXTINDEXES administration table and as the description of the OmniFind Text Search Server for DB2 for i collection.

The value for this option is a string value that is less than or equal to 512 bytes.

IGNOREEMPTYDOCS

Specifies whether to represent empty documents in the text search index. Empty documents are those documents with an empty string or a null value.

The supported values for this option are 0 (zero) and 1. The default value is 1.

If this option is set to 1, empty documents are not represented in the text search index. If you use this option and change the document content to empty, the next incremental update deletes the documents from the text search index.

KEYCOLUMN

Specifies the name of a unique column to be used as the key column in the text index. The key column is used to associate data in the text index to a document or row in the base table. The specified column must have a primary key constraint or unique index. If KEYCOLUMN is not specified, the ROWID column from the table is used, if one exists. Otherwise, the primary key defined on the table is used.

SERVER

Specifies the ID or alias name of the server to be used to store the text search index. If an ID is used, the value is an integer that must exist in the SERVERID column of the QSYS2.SYSTEXTSERVERS catalog. If an alias name is used, the value is a string that must exist in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS catalog. If SERVER is not specified, the default is to select the server with the fewest text search indexes from the servers in the QSYS2.SYSTEXTSERVERS table where parameter SERVERSTATUS is set to 0 (zero), which means that the server is available.

UPDATEAUTOCOMMIT

Specifies how often a commit operation is performed when fetching documents during an index update. A value of 0 (zero) means that a commit operation occurs only at the end of processing.

The value must be an integer between 0 (zero) and 2147483647. The default value is 100.

Performance tip: The value of UPDATEAUTOCOMMIT can have a substantial impact on the performance of index updates. The commit operation that takes place at the specified interval ensures a consistent checkpoint from which to restart the index update, if it is interrupted. However, the commit also temporarily suspends the update process. Increasing the UPDATEAUTOCOMMIT value (or setting it to 0) can substantially improve the update performance, especially the initial update. The value you specify must balance the need for performance with the need for recoverability, based on the frequency of the index updates.

ROW_COLUMN_ACCESS

Specifies whether the text search index is considered secure for row access control and column access control.

NOT_SECURED

Specifies that the text search index is considered not secure for row access control and column access control. This is the default.

The based on table for the index must not have an active permission or Mask.

SECURED

Specifies that the index is considered secure for row access control and column access control.

A text search index must be defined as secured to be built over a table with an active permission or column Mask. If a function is referenced to access or modify data in a masked column, the function must be defined as secured. The authorization ID must have QIBM_DB_SECADM authority to create a text search index with the SECURED attribute.

When a text search index is built over sensitive information there are additional considerations:

The database administrator needs to be aware that the data specified as key column(s) for the text search index will be stored in a staging table in QSYS2 and sent to the text search server using network protocols.

The database administrator needs to be aware that data indexed by a text search index is sent to the text search server using network protocols and stored outside of DB2 on the text search server.

Default values for the *options* parameter

When you install OmniFind Text Search for DB2 for i, the QSYS2.SYSTEXTDEFAULTS table is created and populated with default values for the *options* parameter of the SYSPROC.SYSTS_CREATE stored procedure.

The following table lists the options, default values, and descriptions of the options.

<i>Table 2. Default values for the options parameter</i>		
Option	Default value	Description
CCSID	1208	Specifies the coded character set identifier that is used when binary text documents are indexed.
CJKSEGMENTATION	NGRAM	Specifies the segmentation method to use when you index documents for CJK (Chinese, Japanese, Korean) languages.
LANGUAGE	en_US	Specifies the language used to process text documents.
FORMAT	TEXT	Identifies the format of text documents in the column. The default format is plain text unless the data type is XML.

Table 2. Default values for the options parameter (continued)

Option	Default value	Description
UPDATEFREQUENCY	NONE	Indicates that no automatic updates are scheduled.
UPDATEMINIMUM	1	If at least one document changed since the last index update, the SYSPROC.SYSTS_UPDATE stored procedure starts processing.
IGNOREEMPTYDOCS	1	Specifies that empty documents (documents with an empty string or a null value) are not represented in the text search index. The metadata fields for these documents are not available for search.
UPDATEAUTOCOMMIT	100	Specifies how often a commit operation is performed when documents are fetched during an index update.
MINIMUMUPDATEINTERVAL	5	Specifies the intervals for the UPDATEFREQUENCY option. Intervals cannot be shorter than 5 minutes.
ROW_COLUMN_ACCESS	NOT_SECURED	Specifies whether the index is created as secured or not.
USEREXITTHREADS	0	Reserved

Related concepts

Supported document formats

The text column data can be plain text, an HTML document, an XML document, or any document that is recognized by the search engine.

Supported data types

The data in the text columns that you want to index and search can be either binary data or character data.

Related reference

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

QSYS2.SYSTEXTDEFAULTS administration table

You can see the default parameters and values in the QSYS2.SYSTEXTDEFAULTS administration table. This table is created when you install OmniFind Text Search for DB2 for i.

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

SYSPROC.SYSTS_START

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

Supported languages

You can specify that text documents be processed using a specific language.

SYSPROC.SYSTS_ALTER

You can call the SYSPROC.SYSTS_ALTER stored procedure to modify attributes of an index that was created by SYSPROC.SYSTS_CREATE. Only attributes explicitly specified on this procedure are changed. All other attributes of the index remain unchanged.

This is useful if you need to change the attributes of the index, such as the update frequency, after the index has already been created.

Prerequisites

Before you call the SYSPROC.SYSTS_ALTER stored procedure, verify the following prerequisite:

- The text search index was created (by invocation of the SYSPROC.SYSTS_CREATE stored procedure).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The privilege to create in the schema. For more information, see [Authorization, privileges and object ownership](#).
- Administrative authority

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The following system authorities:
 - *USE to the Create Logical File (CRTLF) command
 - *CHANGE to the data dictionary if the library into which the text index is created is an SQL schema with a data dictionary
- Administrative authority

The privileges held by the authorization ID of the statement must also include at least one of the following privileges:

- For the referenced table:
 - The INDEX privilege on the table
 - The system authority *EXECUTE on the library containing the table
- Administrative authority
- If SQL names are specified, and a user profile exists that has the same name as the library into which the text index is created, and that name is different from the authorization ID of the statement, then the privileges held by the authorization ID of the statement must include at least one of the following authorities:
 - System authority *ADD to the user profile with that name
 - Administrative authority

If a distinct type is referenced, the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- For each distinct type identified in the statement:
 - The USAGE privilege on the distinct type, and
 - The system authority *EXECUTE on the library containing the distinct type
- Administrative authority

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

► SYSTS_ALTER (— *indexSchema* — , — *indexName* — , — *options* —) ►

The schema qualifier is SYSPROC.

Parameters

indexSchema

Identifies the schema of the text search index. If this parameter is null, the value of the CURRENT SCHEMA special register for the invoker is used.

The data type of this parameter is VARCHAR(128).

indexName

Identifies the name of the text search index. The name of the text search index together with the index schema uniquely identifies the text search index in the DB2 subsystem. You must specify a non-null value for this parameter.

The data type of this parameter is VARCHAR(128).

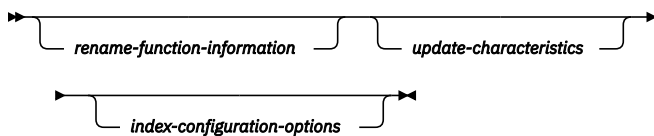
options

A character string that specifies the various options that are available for this stored procedure.

The data type of this parameter is VARCHAR(32000).

The parameter cannot be NULL.

options



rename-function-information

► RENAME FUNCTION — *function-schema* . — *function-name* ►

Specifies the user-defined function to be renamed.

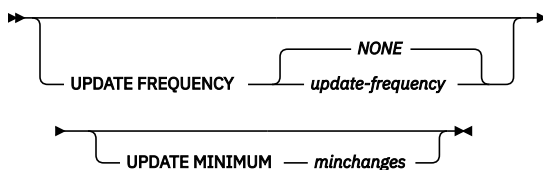
function-schema.function-name

Specifies the schema and name of a user-defined function.

This option is used to change a function that was specified while creating an index. If the function is changed, SYSTS_UPDATE uses the new function to index the text column.

If the function was changed, SYSTS_UPDATE does not change the existing data of the index. Only the new changed data after the last update is processed.

update-characteristics



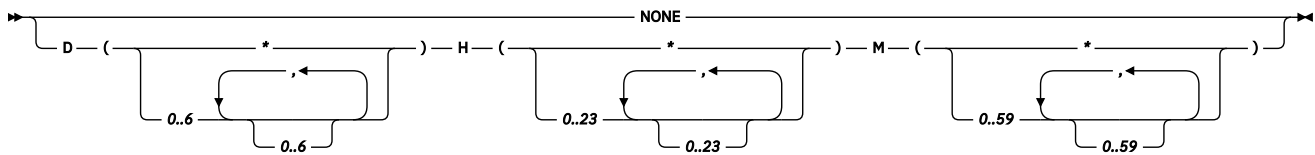
update-characteristics

Specifies the frequency of automatic updates to the text search index. Also specifies the minimum number of changes to text documents before the text search index is updated incrementally at the specified time.

UPDATE FREQUENCY *update-frequency*

Specifies when to make automatic updates to the text search index. The default value is NONE. This option might be useful for a text column in which there are no further changes. The format of the update-frequency option supports two different formats.

update-frequency (Format 1)



NONE

If NONE is specified, then no further index updates are made. The update must be started manually. This option might be useful for a text column in which no further changes are planned.

D

Specifies the day or days of the week when the index is updated. An asterisk (*) specifies all days. 0 specifies Sunday.

H

Specifies the hour or hours when the index is updated. An asterisk (*) specifies all hours.

M

Specifies the minute or minutes when the index is updated. An asterisk (*) cannot be specified. The minimum update frequency is 5 minutes.

Example: This example specifies that the index update is to run every 30 minutes.

```
UPDATE FREQUENCY D(*) H(*) M(0,30)
```

update-frequency (Format 2, chronological)

► <minute> — <hour> — <dayOfMonth> — <monthOfYear> — <dayOfWeek> ►

The format of the *update-frequency (chronological)* option is a list of the five values separated by a blank space. The five values represent the minutes, hours, days of the month, months of the year, and days of the week beginning with Sunday.

If you specify an interval of values or an asterisk (*), you can specify a step value by using a forward slash (/) at the end of the defined interval.

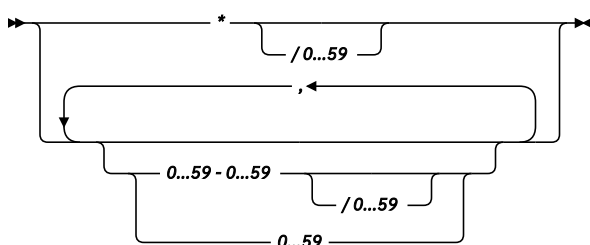
Example: This example specifies that the index update is to run every quarter hour (0, 15, 30, 45) on the even hours between 8 a.m. and 6:45 p.m. (8-18/2 is equivalent to 8, 10, 12, 14, 16, 18), from Monday to Friday every month of the year (* * 1-5).

```
0,15,30,45 8-18/2 * * 1-5
```

minute

Specifies the minutes of the hour when the text search index is to be updated. You can specify an asterisk (*) for an interval of every 5 minutes, or you can specify an integer 0 - 59. You cannot repeat values. The minimum update frequency is 5 minutes. A value of 1,4, or 8 is not allowed.

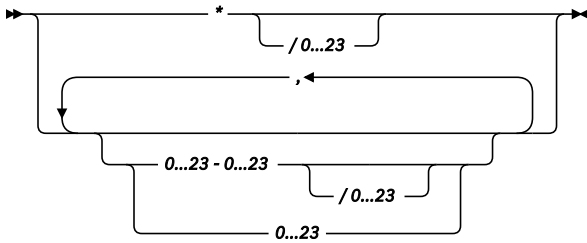
update-frequency (minute)



hour

Specifies the hours of the day when the text search index is to be updated. You can specify an asterisk (*) for every hour, or you can specify an integer 0 - 23. You cannot repeat values.

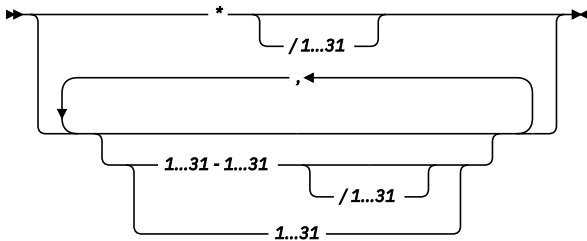
update-frequency (hour)



dayOfMonth

Specifies the days of the month when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 1 - 31. You cannot repeat values.

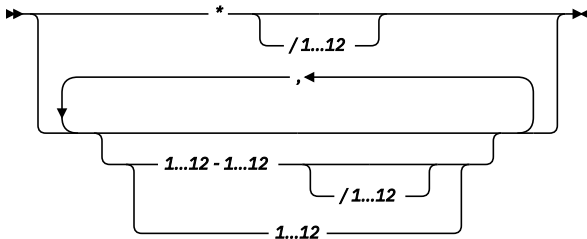
update-frequency (dayOfMonth)



monthOfYear

Specifies the months of the year when the text search index is to be updated. You can specify an asterisk (*) for every month, or you can specify an integer 1 - 12. You cannot repeat values.

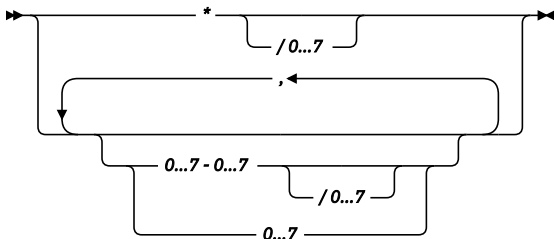
update-frequency (monthOfYear)



dayOfWeek

Specifies the days of the week when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 0 - 7. Both 0 and 7 are valid values for Sunday. You cannot repeat values.

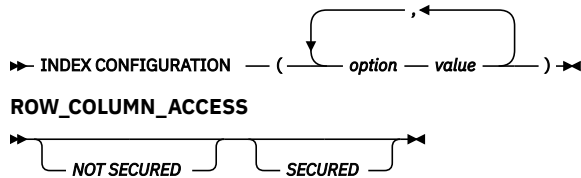
update-frequency (dayOfWeek)



UPDATE MINIMUM *minchanges*

Specifies the minimum number of record changes made to the underlying table before the text search index is updated incrementally at the time specified in the *update-frequency* option. The value must be an integer 1 - 2147483647.

index-configuration-options



COMMENT

Specifies a comment that is stored in the REMARKS column of the QSYS2.SYSTEXTINDEXES administration table and as the description of the OmniFind Text Search Server for DB2 for i collection.

The value for this option is a string value that is less than or equal to 512 bytes.

UPDATEAUTOCOMMIT

Specifies how often a commit operation is performed when fetching documents during an index update. A value of 0 (zero) means that a commit operation occurs only at the end of processing.

The value must be an integer between 0 (zero) and 2147483647.

Performance tip: The value of UPDATEAUTOCOMMIT can have a substantial impact on the performance of index updates. The commit operation that takes place at the specified interval ensures a consistent checkpoint from which to restart the index update, if it is interrupted. However, the commit also temporarily suspends the update process. Increasing the UPDATEAUTOCOMMIT value (or setting it to 0) can substantially improve the update performance, especially the initial update. The value you specify must balance the need for performance with the need for recoverability, based on the frequency of the index updates.

ROW_COLUMN_ACCESS

Specifies whether the text search index is considered secure for row access control and column access control.

NOT_SECURED

Specifies that the text search index is considered not secure for row access control and column access control. This is the default.

The based on table for the index must not have an active permission or Mask.

SECURED

Specifies that the index is considered secure for row access control and column access control.

A text search index must be defined as secured to be built over a table with an active permission or column Mask. If a function is referenced to access or modify data in a masked column, the function must be defined as secured. The authorization ID must have QIBM_DB_SECADM authority to create a text search index with the SECURED attribute.

When a text search index is built over sensitive information there are additional considerations:

The database administrator needs to be aware that the data specified as key column(s) for the text search index will be stored in a staging table in QSYS2 and sent to the text search server using network protocols.

The database administrator needs to be aware that data indexed by a text search index is sent to the text search server using network protocols and stored outside of DB2 on the text search server.

Tips: If users alter an index from SECURED to NOT SECURED, users have to make sure the based table must not have an active permission mask. Or an error will be thrown.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

SYSPROC.SYSTS_DROP

You can call the SYSPROC.SYSTS_DROP stored procedure to drop a text search index that was defined by using the SYSPROC.SYSTS_CREATE stored procedure.

It is recommended that you drop a text search index by using the SYSPROC.SYSTS_DROP stored procedure before dropping the table.

Dropping the view representing the text search index, even as the result of a DROP TABLE CASCADE statement, attempts to drop the text search index. However, because the text search index cannot be dropped under commitment control, the SQL view cannot be dropped under commitment control.

If the text search server cannot be reached, the collection on the server might become orphaned. If that happens, the collection needs to be deleted manually. When the server is available again, use the OmniFind Text Search Server for DB2 for i administration tool to delete the collection on the server.

In [“Administration tools” on page 69](#), you can find information about the tools to identify orphaned indexes and the stored procedure STSPROC.SYSTS_REMOVE or SYSPROC.SYSTS_CLEAR_INDEXES to delete orphaned indexes.

Prerequisites

Before you call the SYSPROC.SYSTS_DROP stored procedure, verify the following prerequisites:

- DB2 text search functions were started by calling the SYSPROC.SYSTS_START stored procedure.
- The text search index was created (by invocation of the SYSPROC.SYSTS_CREATE stored procedure).
- Ensure that the following stored procedures are not running for the text search index that you want to drop: SYSPROC.SYSTS_CREATE, SYSPROC.SYSTS_UPDATE, and SYSPROC.SYSTS_DROP.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The following system authorities:
 - The system authorities of *OBJOPR and *OBJEXIST on the text index to be dropped
 - The system authority *EXECUTE on the library that contains the text index to be dropped
- Administrative authority

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

If an issue occurred while indexing a document from the base table, the staging table column with the TOBEDELETED has a value set to either E (error) or W (warning). ERRORMSG column has value to record error messages. You can retrieve these records with a stored procedure that includes the following query:

```
SELECT s.TOBEDELETED, s.ERRORMSG, based_on_columns
FROM based_on_table t INNER JOIN QSYS2.stagingtables
ON (QQQ_TEXTSEARCH_KEY(t.k1, t.k2, t.k3, ...) = s.KEYID)
WHERE s.TOBEDELETED IN('E','W')
```

In this case, `based_on_columns` is the column list that you need to see from the `based_on_table`. `based_on_table` is the table being indexed. `staging table` is the staging table listed in the catalogs for the text search index. `k1, k2, k3, ...` is the list of key columns in the primary key, row ID, or unique key that is used to build the text search index. `ERRORMSG` is a column records the error messages while indexing this document. Once you correct the errors for those documents, run the update again.

If an index update is requested at the same time an update is already running for that index, an error is returned. Only one update is allowed to run at a time for a given index.

If an index is created as SECURED, the unmasked data is retrieved from the base table and be indexed, like there is no mask or permission created on this table.

If there is field procedure created on the base table, the original unmasked data is indexed.

Prerequisites

Before calling the SYSPROC.SYSTS_UPDATE stored procedure, verify the following prerequisites:

- The text search index was created (by invocation of the SYSPROC.SYSTS_CREATE stored procedure).
- The following stored procedures are not running for the text search index that you want to update: SYSPROC.SYSTS_CREATE, SYSPROC.SYSTS_UPDATE, and SYSPROC.SYSTS_DROP.
- The text search server that the index resides on must have been started using the SYSTS_START stored procedure. The SERVERSTATUS column in QSYS2.SYSTEXTSERVERS must have a value of '0' (started).

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The privilege to create in the schema. For more information, see [Authorization, privileges and object ownership](#).
- Administrative authority

The privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The following system authorities:
 - *USE to the Create Logical File (CRTLF) command
 - *CHANGE to the data dictionary if the library into which the text index is created is an SQL schema with a data dictionary
- Administrative authority

The privileges held by the authorization ID of the statement must also include at least one of the following privileges:

- For the referenced table:
 - The INDEX privilege on the table
 - The system authority *EXECUTE on the library containing the table
- Administrative authority
- If SQL names are specified, and a user profile exists that has the same name as the library into which the text index is created, and that name is different from the authorization ID of the statement, then

the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- The system authority *ADD to the user profile with that name
- Administrative authority

If a distinct type is referenced, the privileges held by the authorization ID of the statement must include at least one of the following privileges:

- For each distinct type identified in the statement:
 - The USAGE privilege on the distinct type, and
 - The system authority *EXECUTE on the library containing the distinct type
- Administrative authority

If the index was created by SYSTS_CREATE as SECURED index, the authorization ID must have DB_SECADMIN authority.

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

➤ SYSTS_UPDATE — (indexSchema , — *indexName* — , — *options* —) ➤

indexSchema can be null.

The schema qualifier is SYSPROC.

Parameters

indexSchema

Identifies the schema of the text search index. If this parameter is null, the value of the CURRENT SCHEMA special register for the invoker is used.

The data type of this parameter is VARCHAR(128).

indexName

Identifies the name of the text search index. The name of the text search index together with the index schema uniquely identifies the full-text index in the DB2 subsystem. You must specify a non-null value for this parameter.

The data type for this parameter is VARCHAR(128).

options

A character string that specifies the option that is available for this stored procedure.

The available option is USING UPDATE MINIMUM. This option uses the USING UPDATE MINIMUM settings that you specified for the SYSPROC.SYSTS_CREATE stored procedure. It starts an incremental update only if the specified number of changes was reached. The default is to unconditionally start the update process.

USING UPDATE MINIMUM

➤ USING UPDATE MINIMUM ➤

Related concepts

[Document truncation](#)

The OmniFind Text Search Server for DB2 for i limits the number of characters that can be indexed for each text document. Sometimes this character limit results in the truncation of large text documents in the text search index.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

SYSPROC.SYSTS_START

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

SYSPROC.SYSTS_DROP

You can call the SYSPROC.SYSTS_DROP stored procedure to drop a text search index that was defined by using the SYSPROC.SYSTS_CREATE stored procedure.

SYSPROC.SYSTS_SHUTDOWN

You can call the SYSPROC.SYSTS_SHUTDOWN stored procedure to shutdown DB2® text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped) and also ends the text search server jobs on the host system.

After this stored procedure has completed, SQL queries that use the CONTAINS or SCORE functions or administration stored procedures used for index maintenance return a failure without trying to contact a text search server.

Changes to the based-on table of the index continue to be logged, even when the server is shutdown. However, scheduled updates of the index do not occur until SYSPROC.SYSTS_START has been invoked.

Authorization

The user ID under which this stored procedure is invoked must have the following privileges:

- *EXECUTE authority on the procedure.
- SELECT and UPDATE privileges on the SYSTEXTSERVERS table.
- *EXECUTE authority on the QSYS2 library of the SYSTEXTSERVERS file.
- *JOBCTL authority or QIBM_DB_SQLADM security special function usage.

For information about the system authorities corresponding to SQL privileges, see [GRANT \(Table or View Privileges\)](#).

Syntax

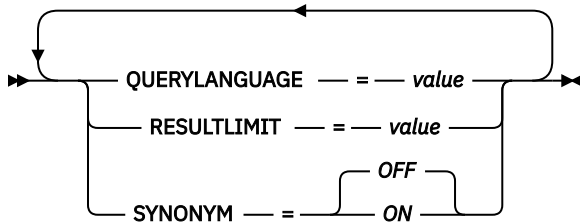
```
>>-SYSTS_SHUTDOWN--(--+-----+--)------><
                    +-serverid--+
                    '-aliasname-'
```

Parameters

serverid or aliasname

Specifies the identifier of the server for clear orphaned indexes. A *serverid* or server *aliasname* is a string. If no identifier is provided, the default is to clear orphaned indexes on all servers. The identifier string must either be a valid *serverid* that exists in the SERVERID column, or a valid server aliasname that exists in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS table. If the identifier can be converted to an integer value, it is interpreted as a *serverid*. If the identifier cannot be converted to an integer value, it is interpreted as a server *aliasname*.

The data type of this parameter is VARCHAR(128).



1

Notes:

¹ The same clause must not be specified more than once.

The schema is QSYS2.

column-name

Specifies a qualified or unqualified name of a column that has a text search index that is to be searched. The column must exist in the table or view that is identified in the FROM clause of the statement. The column of the table, or the column of the underlying base table of the view, must have an associated text search index (SQLSTATE 38H12). The underlying expression of the column of a view must be a simple column reference to the column of an underlying table, directly or through another nested view.

search-argument

Specifies an expression that returns a string value containing the terms used in the search. The expression must not be all blanks or the empty string (SQLSTATE 38H14). The actual length of the string must not exceed 32704 bytes. This length might be further limited by what is supported by the text search server (SQLSTATE 38H10). The value is converted to Unicode before it is used to search the text search index.

string-constant

Identifies a string constant that specifies the search argument options that are in effect for the function.

The options that can be specified as part of the *search-argument-options* are as follows:

QUERYLANGUAGE = value

Specifies the query language. The value can be any of the supported language codes. If the QUERYLANGUAGE option is not specified, the default is the language value of the text search index that is used when this function is invoked. If the language value of the text search index is AUTO, the default value for QUERYLANGUAGE is en_US.

RESULTLIMIT = value

Specifies the maximum number of results to be returned from the underlying search engine. The *value* can be an integer value 1 - 2 147 483 647. If the RESULTLIMIT option is not specified, no result limit is in effect for the query.

This scalar function might not be called for each row of the result table, depending on the plan that the optimizer chooses. This function can be called once for the entire query to the underlying search engine. A result set of all the primary keys that match are returned. This result set is then joined to the table containing the column to identify the result rows. In this case, the RESULTLIMIT value acts like FETCH FIRST ?? ROWS from the underlying text search engine, and can be used as an optimization. If the search engine is called for each row of the result because that is the best plan, then the RESULTLIMIT option is ignored.

SYNONYM = OFF or SYNONYM = ON

Specifies whether to use a synonym dictionary that is associated with the text search index. You can add a synonym dictionary to a collection by using the synonym tool.

OFF

OFF is the default value.

ON

Use the synonym dictionary that is associated with the text search index.

The result of the function is a large integer. If the second argument can be null, the result can be null. If the second argument is null, the result is the null value.

The result is 1 if the document contains a match for the search criteria that are specified in the search argument. Otherwise, the result is 0. The result is also 0 if the column is null. If the search argument is Null, then the result is the null value.

CONTAINS is a nondeterministic function.

Example 1

The following statement finds all the employees who have "COBOL" in their resume.

```
SELECT EMPNO
FROM EMP_RESUME
WHERE RESUME_FORMAT = 'ascii'
AND CONTAINS(RESUME, 'COBOL') = 1
```

Example 2

The search argument does not need to be a string constant. The search argument can be any SQL string expression, including a string contained in a host variable. The following statement searches for the exact term "ate" in the COMMENT column.

Note: The term "ate" must be delimited in double quotes so that only the exact term is searched for and linguistic variations are not considered.

```
char search_arg[100]; /* input host variable */
...
EXEC SQL DECLARE C3 CURSOR FOR
SELECT CUSTKEY
FROM K55ADMIN.CUSTOMERS
WHERE CONTAINS(COMMENT, :search_arg)= 1
ORDER BY CUSTKEY;
strcpy(search_arg, "\"ate\"");
EXEC SQL OPEN C3;
...
```

Example 3

The following statement finds 10 students at random who wrote online essays that contain the phrase "fossil fuel" in Spanish, which is "combustible fósil." These students are for a radio interview. Use the synonym dictionary that was created for the associated text search index. Because only 10 students are needed, optimize the query by using the RESULTLIMIT option to limit the number of results from the underlying text search server.

```
SELECT FIRSTNAME, LASTNAME
FROM STUDENT_ESSAYS
WHERE CONTAINS(TERM_PAPER, 'combustible fósil',
'QUERYLANGUAGE= es_ES RESULTLIMIT = 10 SYNONYM=ON') = 1
```

Related tasks

[Search a text search index](#)

You can search a text search index by using an SQL statement with a CONTAINS or SCORE function. The search argument criteria is specified on the function.

Related reference

[SCORE](#)

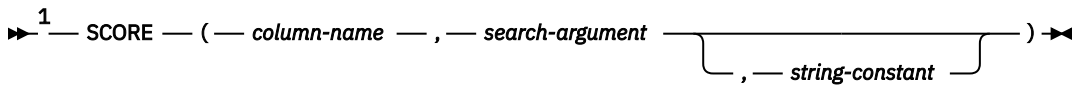
You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

[Search argument syntax](#)

You can specify a search argument as the condition of a search for terms in text documents. It consists of search parameters and one or more search terms. The SQL scalar text search functions that use search arguments are CONTAINS and SCORE.

SCORE

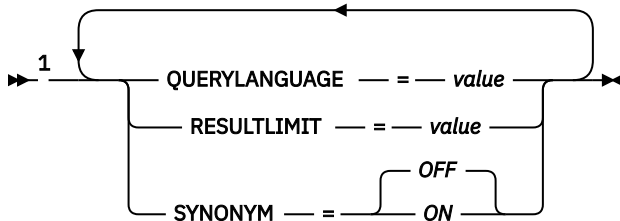
You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.



Notes:

¹ *string-constant* must conform to the rules for the *search-argument* options.

search-argument-options



Notes:

¹ The same clause must not be specified more than once.

The schema is QSYS2.

column-name

Specifies a qualified or unqualified name of a column that has a text search index that is to be searched. The column must exist in the table or view that is identified in the FROM clause of the statement. The column of the table, or the column of the underlying base table of the view, must have an associated text search index (SQLSTATE 38H12). The underlying expression of the column of a view must be a simple column reference to the column of an underlying table, either directly or through another nested view.

search-argument

Specifies an expression that returns a string value containing the terms used in the search. The expression must not be all blanks or the empty string (SQLSTATE 38H14). The actual length of the string must not exceed 32704 bytes. This length might be further limited by what is supported by the text search server (SQLSTATE 38H10). The value is converted to Unicode before it is used to search the text search index. If the search-argument is null, the result is the null value.

string-constant

Identifies a string constant that specifies the search argument options that are in effect for the function.

The options that can be specified as part of the *search-argument-options* are as follows:

QUERYLANGUAGE = value

Specifies the query language. The value can be any of the supported language codes. If the QUERYLANGUAGE option is not specified, the default is the language value of the text search index that is used when this function is invoked. If the language value of the text search index is AUTO, the default value for QUERYLANGUAGE is en_US.

RESULTLIMIT = value

Specifies the maximum number of results that are to be returned from the underlying search engine. The *value* can be an integer value 1 - 2 147 483 647. If the RESULTLIMIT option is not specified, no result limit is in effect for the query.

This scalar function might not be called for each row of the result table, depending on the plan that the optimizer chooses. This function can be called once for the entire query to the underlying search engine. A result set of all the primary keys that match are returned. This result set is then joined to the table containing the column to identify the result rows. In this case, the RESULTLIMIT value acts like FETCH FIRST ?? ROWS from the underlying text search engine and can be used as an optimization. If the search engine is called for each row of the result because that is the best plan, then the RESULTLIMIT option is ignored.

SYNONYM = OFF or SYNONYM = ON

Specifies whether to use a synonym dictionary that is associated with the text search index. You can add a synonym dictionary to a collection by using the synonym tool.

OFF

OFF is the default value.

ON

Use the synonym dictionary that is associated with the text search index.

The result of the function is a double-precision floating-point number. If the second argument can be null, the result can be null. If the second argument is null, the result is the null value.

The result is greater than 0 but less than 1 if the column contains a match for the search criteria that the search argument specifies. The more frequently a match is found, the larger the result value. If the column does not contain a match, the result is 0. The score is also 0 if the column is null.

SCORE is a nondeterministic function.

Example

The following statement generates a list of employees ordered by how well their resumes match programmer AND (java OR cobol). In addition, a relevance value that is normalized between 0 (zero) and 100 is returned.

```
SELECT EMPNO, INTEGER(SCORE(RESUME, 'programmer AND
(java OR cobol)') * 100) AS RELEVANCE
FROM EMP_RESUME
WHERE RESUME_FORMAT = 'ascii'
ORDER BY RELEVANCE DESC
```

Related tasks

[Search a text search index](#)

You can search a text search index by using an SQL statement with a CONTAINS or SCORE function. The search argument criteria is specified on the function.

Related reference

[CONTAINS](#)

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

[SYSPROC.SYSTS_CREATE](#)

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

[Search argument syntax](#)

You can specify a search argument as the condition of a search for terms in text documents. It consists of search parameters and one or more search terms. The SQL scalar text search functions that use search arguments are CONTAINS and SCORE.

Search argument syntax

You can specify a search argument as the condition of a search for terms in text documents. It consists of search parameters and one or more search terms. The SQL scalar text search functions that use search arguments are CONTAINS and SCORE.

For any language-specific processing during a search, you can specify a value for the QUERYLANGUAGE parameter as a search argument option. The value can be any of the supported language codes.

If the QUERYLANGUAGE parameter is not specified, the default value is the language value of the text search index used when this function is called.

If the language value of the text search index is AUTO, the default value for QUERYLANGUAGE is en_US.

Limitations

You cannot use the CONTAINS and SCORE functions in an SQL constraint or index definition. You can use them in SQL query statements and view definitions under the following restrictions:

- If a view, nested table expression, or common table expression provides a text search column for a CONTAINS or SCORE scalar function, and if the applicable view, nested table expression, or common table expression has a DISTINCT clause on the outermost SELECT statement, then the SELECT list must contain all the corresponding key fields of the text search index. Otherwise, SQL message 38H12 is returned.
- If a view, nested table expression, or common table expression provides a text search column for a CONTAINS or SCORE scalar function, then the applicable view, nested table expression, or common table expression cannot have a UNION, an EXCEPT, or an INTERSECT statement at the outermost SELECT level. Otherwise, SQL message 38H12 is returned.
- If a common table expression provides a text search column for a CONTAINS or SCORE scalar function, the common table expression can be referenced again in the entire query only when the reference does not provide a text search column for a CONTAINS or SCORE scalar function. Otherwise, SQL message 38H12 is returned.
- A function cannot be created sourced on the CONTAINS or SCORE scalar functions. Otherwise, SQL message SQL0457 is returned.
- The query can run through the SQL Query Engine (SQE).

Simple search

To do a simple keyword search, enter one or more keywords in the query. The search engine returns documents that contain all those keywords, or variations of the keywords.

For example, if you enter `king`, the search engine returns all documents that contain the word `king` or `kings`. If you enter the query `king lear`, the search engine returns documents that contain the terms `king` and `lear`.

To see more precise results, use more specific keywords. For example, use `French roast coffee` rather than `coffee`, or use `Kauai hiking tours` rather than `Hawaiian vacations`.

If a simple keyword search returns too many documents that are not what you are looking for, you can use operators to refine your search.

Exclusion of terms in a search

Use the minus sign (-) to exclude terms. For example, if you want to find documents with the term `lear`, and not `edward`, enter the query `lear -edward`.

The minus sign (-) also applies to a term and its variants. For example, the query -edward excludes documents that contain the word edward 's.

Phrase search

If you want to ensure that terms are displayed exactly in the sequence that you typed them in, you can use double quotation marks. For example, if you want to see documents with the term king lear exactly, and not related phrases such as kingly lear or king and queen lear, enter "king lear". The search is not case-sensitive, but term variants are not considered matches.

Language processing

OmniFind Text Search performs language-specific processing on terms by using the language that is specified by the query. When the language is not specified, the default language is used.

When you search for a word, the base form of the word is also searched. For example, searching for tests or testing also finds the word test.

During language processing, predefined synonyms are added to the query. Language processing is not performed on phrases and on terms in all capital letters, for example, DOG.

Wildcard character in a search

The wildcard character (*) helps you find documents when you do not know the full spelling, or if you want to find variations of the term. For example, the query czech* returns documents with the terms czech, czechoslovakia, czechoslovakian, czech republic, and other possible results.

You can also use the wildcard character in a phrase search. For example, the query "John * Kennedy" returns documents with the terms John Fitzgerald Kennedy and John F Kennedy but not John Kennedy. The query Mi*l Gorbachev returns Mikhail Gorbachev.

Adding a wildcard character to the beginning of a query (for example, *zech) might cause the search engine to take longer to return results.

Searches for at least one of the terms

The logical operator OR specifies that at least one of the terms in a query must be displayed in the returned document. For example, the query (othello OR otello) returns documents that contain the term othell or otello.

You can also use the logical operators AND, OR, and NOT in combinations by using parentheses. For example, the query cougar OR (jaguar AND NOT car) returns documents with the terms cougar or jaguar but not car.

You must enter the logical operators AND, OR, and NOT in all uppercase. Use parentheses for grouping.

Related concepts

XML search

You can index and search XML documents. The XML search grammar uses a subset of the W3 XPath language with extensions for text search. The extensions support range searches of numeric, Date, and DateTime values that are associated with an XML attribute or element. Structural elements can be used separately, or combined with free text in queries.

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Simple query examples

Simple queries for the CONTAINS and SCORE functions search for a single word or multiple words in a text search index.

The search engine ignores white space between characters. The search string must not be empty or contain all blanks (SQLSTATE 38H14).

The following table shows some examples of simple search queries.

Table 3. Simple query examples

Search word types	Examples	Query results
Single word	king	Returns all documents that contain the word <code>king</code> or <code>kings</code> . This query matches different surface forms and is not case sensitive.
Multiple words	king lear	Returns all documents that contain <code>king</code> and <code>lear</code> . The default operator is the logical operator AND.

The operators **AND** and **+** are implicit in every query. For example, the query `King Lear` returns the same results as `King AND Lear` or `King + Lear`.

You must enter the logical operators **NOT**, **AND**, and **OR** in all uppercase.

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Advanced search operators

You can use advanced search operators to refine the search results for the CONTAINS function and the SCORE function.

In the following table, the first column describes the operator that you can use in a search query. You must enter the logical operators NOT, AND, and OR in all uppercase letters. The second column shows a sample query that you might enter. The third column describes the types of results that you might see from the example query.

Table 4. Advanced search operators and complex query examples

Operators	Examples	Query results
AND	<pre>"King Lear" AND "Othello"</pre> <pre>"King Lear" "Othello"</pre>	Either query returns documents that contain both terms <code>King Lear</code> and <code>Othello</code> . The AND operator is the default conjunction operator. If no logical operator is between the two terms, the AND operator is used. For example, the query <code>King Lear</code> is the same as the query <code>King AND Lear</code> .

Table 4. Advanced search operators and complex query examples (continued)

Operators	Examples	Query results
OR	"King Lear" OR Lear	Returns documents that contain either King Lear or just Lear. The OR operator links the two terms and finds a matching document if either of the terms exist in a document.
NOT	"King Lear" NOT "Norman Lear"	Returns documents that contain King Lear but not Norman Lear.
" " (Exact match)	First query: "King Lear" Second query: "king"	The first query returns the exact phrase King Lear. The second query returns only the word king and no other forms, such as kings or kingly.
* (Wildcard character)	test* test	Returns documents that can match possible combinations, such as test, tests, and tester, or test and text.
^ (Score boost factor) <i>some word or phrase^number</i>	First query: "King Lear"^4 "Richard III" Second query: title: (software download)^5 pdf viewer -shipping	The first query forces documents with the phrase King Lear to be displayed higher in the list of search results. The second query forces a document titled software download to be displayed higher in the list of results. Although a boost factor must be positive, the boost factor can be less than 1. For example, 0.2. The boost factor number has no limit.
+ (Includes)	+Lear King	Returns all documents that contain Lear and King, which is the same as the query Lear AND King.
- (Excludes)	"King Lear" - "Lear Jet"	Returns documents that contain King Lear but not Lear Jet.
()	(King OR Lear) AND plays	Returns documents that contain either King or Lear and plays. The parentheses ensure that plays is found and either term King or Lear is present.

Table 4. Advanced search operators and complex query examples (continued)

Operators	Examples	Query results
<p>\</p> <p>(Escape character)</p>	<pre>\(1+1\) :2</pre>	<p>Returns documents that contain (1+1) :2. Use the \ as an escape character so that you can use special characters that are normally part of the query syntax. If a special character is preceded by the escape character, the special character is analyzed as part of the query. Special characters are: +, -, &&, , !, (,), {, }, [,], ^, ", ~, *, ?, :, and \. If a special character is cleared, the special character is analyzed as part of the query.</p>
<p>%</p> <p>Optional terms</p>	<pre>log %file</pre>	<p>Returns documents that include the term <code>log</code> and optionally include the term <code>file</code>.</p>
<p>~</p> <p>Fuzzy Search</p>	<pre>analytics~ analytics~0.8</pre>	<p>The first query returns documents that include the terms <code>analytics</code>, <code>analyze</code>, <code>analysis</code>, and so on.</p> <p>A fuzzy search query searches for character sequences that are not only the same but similar to the query term. Use the tilde symbol (~) at the end of a term to do a fuzzy search.</p> <p>You can add an optional parameter to specify the required similarity. Specify a value greater than 0 and less than 1. The value must be preceded by a 0 and decimal point, for example, 0.8. A value closer to 1 matches terms with a higher similarity. If the parameter is not specified, the default is 0.5.</p> <p>Restriction: Special characters are not supported in proximity search queries.</p>
<p>~</p> <p>Proximity searches</p>	<pre>"IBM WebSphere"~7</pre>	<p>Returns documents that contain "IBM" and "WebSphere" within seven words of each other.</p> <p>A proximity search finds documents that contain terms within a specified number of words of each other. Use the tilde symbol (~) to do a proximity search.</p> <p>Proximity search is supported for individual terms, not for phrases. Also note that a word after a sentence break is not considered adjacent to words in the previous sentence.</p> <p>Restriction: Special characters are not supported in proximity search queries.</p>

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

Searching for special characters

OmniFind supports indexing and searching special characters.

You can search for special characters like other query terms. To find a special character in a document, include the special character in the query expression. In some cases, escaping special characters is required.

Escaping special characters

Special characters can serve different functions in the query syntax. For example, question marks (?) can be used as wildcard characters. To search for a special character that has a special function in the query syntax, you must escape the special character by adding a backslash before it, for example:

- To search for the string “where?”, escape the question mark as follows: “where\?”
- To search for the string “c:\temp,” escape the colon and backslash as follows: “c:\\temp”

Not escaping such special characters can result in syntax errors.

Table 5. Special characters that must be escaped to be searched

Special Character	Notes on behavior when not escaped
Ampersand (&)	
Asterisk(*)	Used as a wildcard character.
At sign (@)	A syntax error is generated when an at sign is the first character of a query. In xmlxp expressions, the at sign is used to refer to an attribute.
Brackets []	Used in xmlxp expressions to search the contents of elements and attributes
Braces { }	Generates a syntax error.
Backslash (\)	
Caret (^)	Used for weighting (boosting) terms.
Colon (:)	Used to search in the contents of fields.
Equal sign (=)	Generates a syntax error.
Exclamation point (!)	A syntax error is returned when an exclamation point is the first character of a query.
Forward slash (/)	In xmlxp expressions, a forward slash is used as an element path separator.
Greater than symbol (>) Less than symbol (<)	Used in xmlxp expressions to compare the value of an attribute. Otherwise, these characters generate syntax errors.

Table 5. Special characters that must be escaped to be searched (continued)

Special Character	Notes on behavior when not escaped
Minus sign (-)	When a minus sign is the first character of a term, only documents that do not contain the term are returned.
Parentheses ()	Used for grouping.
Percent sign (%)	Specifies that a search term is optional.
Plus sign (+)	
Question mark (?)	Handled as a wildcard character.
Semicolon (;)	
Single quotation mark (')	Single quotation marks are used to contain xmlxp expressions.
Tilda (~)	Handled as proximity and fuzzy search operators
Vertical bar ()	

Escaping special characters that do not serve a special function in the query syntax is optional. The following table shows some examples of special characters that do not require escaping.

Table 6. Examples of special characters that do not require escaping

Special Character	Notes on behavior when not escaped
Comma (,)	
Dollar sign (\$)	
Period (.)	In xmlxp expressions, a period is used to search the content of elements.
Pound sign (#)	
Underscore (_)	

Special characters adjacent to query terms

When a special character is adjacent to a word in a query, documents that contain the special character and word in the same order are returned. For example, searching for “30\$” finds documents that contain “30\$”, but does not find documents that contain “\$30”. However, searching for “30 \$” (with a space) finds all documents that contain “30” and “\$” anywhere in the documents including both “30\$” and “\$30”.

When a special character is adjacent to a stop word in a query, the stop word is not removed from the query. For example, searching for “at&t” does not remove the stop word “at”. However, searching for “at & t” with spaces removes the stop word “at”.

When a special character separates two words, the sequence of tokens is searched as a sequence. For example, searching for “jack_jones” finds documents that contain “jack_jones” but not documents that contain “jack_and_jones”.

Words that are adjacent to special characters are lemmatized. For example, searching for “cats&dogs” in English finds documents that contain “cat&dog”.

You can use special characters in wildcard search expressions. For example, searching for “ja*_” finds documents that contain “jack_jones”. However, you cannot use wildcard characters to find special characters. For example, searching for “ca*s” finds documents that contain “cats”, “categories”, or “cas”, but not documents that contain “ca_s”.

Indexing special characters

During tokenization and language processing, OmniFind server identifies and indexes special characters as punctuation. Special characters are token delimiters.

For example, “jack_jones” is tokenized as three separate tokens: “jack”, “_”, and “jones”. Emails, URLs, and file paths are broken down into tokens, for example:

- Jack_jones@ibm.com is tokenized as jack _ jones @ ibm . com
- http://www.ibm.com is tokenized as http :// www . ibm . com

Special characters do not occupy a token position in the file. For example, "jack_jones" is indexed with the underscore in the same token position as "jack". Special characters also do not occupy a token position when spaces are included. For example, “jack_jones” is indexed in the same way as “jack _ jones”.

The token position is used for exact phrase search and for proximity search. For example, if a document contains the expression jack_jones, searching for the exact phrase ““jack jones”” finds this document.

When a sequence of special characters are indexed separately, they are searched in no particular order. For example, searching for “#\$” also finds documents that contain “\$#”.

Special characters in CJK languages

To find a sequence of characters that includes special characters, the query expression must include the special characters. If you omit the special characters from the query expression, the character sequence might not be found. In non-CJK languages, the character sequence is always found, even if the query expression omits the special characters. For example, if an indexed document contains john_smith, you can search for john_smith or "john smith" (exact match, without the underscore) and both queries return the document that contains john_smith.

Restriction: You cannot search for the following special characters in CJK documents: ? * \

Example using CONTAINS and SCORE functions

You can use the CONTAINS and SCORE functions in the same query. The query searches a text search index and return if and how frequently the text document matches the search argument criteria.

The example in the following table uses data from the base table BOOKS with the columns ISBN (VARCHAR(20)), ABSTRACT (VARCHAR(10000)), and PRICE (INTEGER).

Table 7. The base table BOOKS

ISBN	ABSTRACT	PRICE
i1	"a b c"	7
i2	"a b d"	10
i3	"a e a"	8

You run the following query:

```
SELECT ISBN, SCORE(ABSTRACT, '"b"')
FROM BOOKS
WHERE CONTAINS (ABSTRACT, '"b"') = 1
```

This query returns the following two rows:

```
i1,    0.3
i3,    0.4
```

The score values might differ depending on the content of the text column.

Related reference

[CONTAINS](#)

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

XML search

You can index and search XML documents. The XML search grammar uses a subset of the W3 XPath language with extensions for text search. The extensions support range searches of numeric, Date, and DateTime values that are associated with an XML attribute or element. Structural elements can be used separately, or combined with free text in queries.

Documents must be indexed to include the XML markup before the index can be searched using the xmlxp query syntax. Document indexing is done by using the “FORMAT XML” option at index creation time.

Indexes created on a previous release can be used to perform searches. However, documents indexed on a previous release do not have the information necessary to use all the XML search capabilities available in a newer release. Documents added or updated in the text search index after the upgrade to the new release include the additional information.

An upgrade might result in documents indexed on the prior release not being included in some search results. The SYSPROC.SYSTS_REPRIMEINDEX stored procedure can be used to rebuild the index and resolve this problem.

To use the OMNIFIND CONTAINS and SCORE built-in functions to search XML data, the query string must start with the @xmlxp: query prefix. The prefix is followed by a valid XML Search query expression. The @xmlxp 'opaque' term prefix indicates that a search is performed using the query path expression.

For example: CONTAINS(columnname, '@xmlxp:''query_expression''').

The single quotes ' ' surrounding the query_expression must be doubled because they are contained within an SQL string, in effect, a string within a string.

The @xpath: opaque term prefix that was used in previous releases of OmniFind Text Search Server for DB2 for i is supported for compatibility with earlier versions. However, it has been deprecated and is not recommended.

The following list highlights the key features of XML search:

XML structural search

By including special opaque XML terms in queries, you can search XML documents for structural elements and text that is scoped by those elements. Structural elements are tag names, attribute names, and attribute values. Element and tag names are case sensitive.

XML query tokenization

Tokenization is the process of parsing input into tokens. Free text in XML query terms is tokenized the same way that text in non-XML query terms is tokenized. An exception is that nested opaque terms are not supported. Free text search is not case sensitive.

XML Schema and DTD

Any XML schema associated with the XML document is not downloaded, and default values are not indexed.

Numeric values

Predicates that compare attribute or element values to numbers are supported.

Element values

Predicates that compare element values to numbers or dates are supported. The element containing the date or number must be an XML element that contains only the number or date. Leading and trailing white space are ignored.

String values

Use of the = operator for a string argument in a predicate requires a complete match of all key words in the string with tokens in the identified text span. The order of the tokens is not significant when matching is performed.

DateTime values

Predicates that compare Date or DateTime attributes or elements are supported.

Path expressions:

<i>Table 8. Path expressions</i>	
@xmlxp Expression	Description
TagName	Selects a tag named TagName, and all children of that tag.
@AttributeName	Selects an attribute named @AttributeName.
/	Selects from root node.
//	Selects matching tags and attributes that are descendants of the current position and match the expression.
.	Self: the current tag or element node.

<i>Table 9. Path expression examples:</i>	
@xmlxp Expression	Result
/Document	Returns all documents with a top-level tag Document.
//Document	Returns all documents with a tag Document at any level.
/Document/Child1	Returns all documents with a top-level tag Document that has a direct child tag Child1.
/Document//Child1	Returns all documents with a top-level tag Document that has a descendant tag Child1 at any level.
/Root/@attr1	Returns all document with a top-level tag Root with an attribute attr1.
/Root//@attr1	Returns all documents with a top-level tag Root with an attribute attr1 on that root tag or any descendant tag.
//@attr1	Returns all documents that have an attribute @attr1 at any level.

Note: The XML Search expression must have an actual tag or attribute name in the relative path expression. / and // by themselves are not valid search queries.

Path expressions are only allowed in the forward direction, and only on a single axis.

It is recommended that a path expression start with a leading / or //. This indicates that the expression's initial context is the document's root node. When the leading / or // is omitted, the expression is matched at any level. In other words, 'Sentences' is treated as '//Sentences'. The behavior is defined this way to be compatible with prior releases, and does not follow the W3 or SQL/XML standard.

Path expression wildcard support

In the path expression, the special wild-card character * can be used to indicate exactly one tag, with any name.

Trailing path expression wildcards are ignored.

The following uses of path expression wildcards are not supported and result in an error:

- An expression that references only wildcards and no specific elements or attributes.
- A wildcard attribute at any level: /Tag/@*.
- A wildcard that immediately precedes a predicate expression: /Root/*[//anytag].
- A wildcard that is used in a predicate comparison: /Root[* > 5].
- A wildcard as an XML namespace prefix: //*:tagname.
- A wildcard prefixed with an XML namespace prefix: //ns:*
- A wildcard character used as part of a tag name: /start*.

<i>Table 10. Path expression wildcard examples:</i>	
@xmlxp Expression	Result
/Root/*/T1	All documents having a top-level tag Root that has a descendant tag T1 with one intermediate level.
/Root*//T1	All documents having a top-level tag Root that has a descendant tag T1 with one or more intermediate levels.

Predicates

Predicates are used to specify a value or condition that an element or attribute node must satisfy. Predicates are always enclosed in square brackets: [].

<i>Table 11. Predicate examples:</i>	
@xmlxp Expression	Result
/Book[Sentences]	Top-level tag is Book and must have a direct child Sentences.
/Book[.//Sentences and .//Author]	Top-level tag is Book and must have both Sentences and Author descendants. Each descendant can be at any level below Book.

Because path expressions are always in the forward direction, and limited to a single access, path expressions in predicates must be relative to the current node. /Book[/Root] and /Book[//Root] are not valid, because in both cases the predicate path expression begins with the top-level tag 'Root' instead of the current node.

Numeric comparisons

OMNIFIND supports the =, <=, >=, >, <, and != operators for comparisons of elements and attributes to integers and floating point values.

Elements have only their numeric values indexed if they are simple elements. They must not contain additional characters (other than white space) and must not have any descendant elements. Complex elements are indexed as text only.

<i>Table 12. Numeric comparison examples:</i>	
@xmlxp Expression	Result
/Book[@id_num = 12345]	Top-level tag is Book and must have an attribute id_num with a value of 12345.
/Book[Cost <= 100.50]	Top-level tag is Book. Book has a direct child element Cost with a numeric value less than or equal to 100.50.

Date and DateTime comparisons

OMNIFIND supports the =, <=, >=, >, <, and != operators for comparisons of elements and attributes to Date and DateTime values.

Simple elements have only their DateTime values indexed. These elements must not contain additional characters (other than white space) and must not have any descendant elements. Complex elements are indexed as text only.

During indexing, attribute values and text contained within simple XML tags are examined. If the text is determined to match an ISO Date or DateTime format, it is indexed as a Date or DateTime that can be searched in a predicate.

During a search, the Date or DateTime value must be enclosed within an xs:date() or xs:dateTime() function call in order to be recognized as the correct data type.

An XML DateTime data type in an XML document can specify a timezone value. However, when a DateTime is indexed, the Text Search server truncates timezone values during indexing. Therefore, timezones are not considered during XML searches that involve Date or DateTime data types.

In addition, a DateTime with an hour of 24 is permitted only if the minutes and seconds are zero. It will be treated as a value between the last instant of that day and the first instant of the next day.

When a value Date or DateTime is specified in an XML search predicate, a syntax error occurs if a time zone is specified on the value.

The DateTime data type supports up to 12 digits of fractional seconds.

<i>Table 13. Date and DateTime comparison examples:</i>	
@xmlxp Expression	Result
/Book[@publishDate > xs:date("2000-01-01")]	Top-level tag is Book. Book has an attribute publishDate that is greater than the date of 2000-01-01.
/Book[purchaseTime > xs:dateTime("2009-05-20T13:00:00")]	Top-level tag is Book. Book has a direct child purchaseTime that is a DateTime expression greater than 2009-05-20T13:00:00.000000.

Contains and excludes in XML markup

The *contains* and *excludes* functions are used to perform full text searches within the XML markup. *Contains* returns true if the query is contained within the target node; *excludes* returns true if the query is NOT contained within the target node.

For example, find all documents with a top-level tag called `email`, and a direct descendant called `body` that contains variations of the phrase “Department budget”.

```
@xnkxo: '/email[body contains ("department budget")]'
```

The free text passed to the *contains* or *excludes* function is handled in the same way as any other free text search. The search is not case-sensitive, and linguistic variations are considered. The earlier query matches “departments budgets” and also “budget for the department”.

The search can be restricted to an exact match by using the traditional quotation marks, for example, `@xmlxp: '/email[body contains("department budget")]'`. The quotes indicating an exact match must be doubled so that they are not interpreted as the end of the *contains* free text string.

Table 14. Contains and excludes examples:	
@xmlxp Expression	Result
<code>/Book[abstract contains("cat AND dog")]</code>	Top-level tag <code>Book</code> that has a child tag <code>abstract</code> which contains linguistic variations of the terms <code>cat</code> and <code>dog</code> .
<code>/Book[abstract contains("cat AND dog")] /Book/@title[. contains("cat OR dog")]</code>	Top-level tag <code>Book</code> has an attribute <code>title</code> that contains linguistic variations of either <code>cat</code> or <code>dog</code> .
<code>/Book/Title[. contains("All good dogs go to heaven")]</code>	Top-level tag <code>Book</code> with a direct child <code>Title</code> that contains all good dogs go to heaven in order, and without linguistic variations being considered.
<code>/Book[abstract excludes("cat AND dog")]</code>	Top-level tag <code>Book</code> that has a child tag <code>abstract</code> which does not contain linguistic variations of the terms <code>cat</code> and <code>dog</code> .

Complete string match operator

The `=` operator with a string argument in a predicate calls for a complete match of all tokens in the string with all tokens in the identified text span. Linguistic equivalents are not considered. The order of the terms searched for is not significant. It is not required that the element or attribute contain only the text that was searched for.

Table 15. Complete string match operator examples:	
@xmlxp Expression	Result
<code>/Book[@author = "Nicholas Lawrence"]</code>	Top-level tag <code>Book</code> that has an attribute <code>author</code> . <code>author</code> must contain the terms <code>Nicholas Lawrence</code> . Linguistic variations on those terms are not considered matches.
<code>/Book[author = "Nicholas Lawrence"]</code>	Top-level tag <code>Book</code> that has a direct descendant <code>author</code> . <code>author</code> must contain the terms <code>Nicholas Lawrence</code> in order. Linguistic variations on those terms are not considered matches.

Logical Operators

The logical operators *and* and *or* can be used in predicates.

Table 16. Logical operator examples:

@xmlxp Expression	Result
/Book[@author = ""Nicholas Lawrence""]/Price[. < 1000 and @unit = "dollars"]	<p>Top-level tag Book that has an attribute author. author must contain the terms Nicholas Lawrence in order. Linguistic variations on those terms are not considered matches.</p> <p>Book must have a direct child Price with a value < 1000. The Price node must have an attribute @unit that has a value of dollars.</p>

Operator precedence

In XML search predicates, containment operators and comparison operators take precedence over logical operators, and all logical operators have the same precedence.

- Containment operators are *contains* and *excludes*.
- Comparison operators are =, !=, <, >, <= and >=.
- Logical operators are *and* and *or*.

You can use parentheses to ensure the precedence that you want.

Related reference

Search argument syntax

You can specify a search argument as the condition of a search for terms in text documents. It consists of search parameters and one or more search terms. The SQL scalar text search functions that use search arguments are CONTAINS and SCORE.

SYSPROC.SYSTS_REPRIMEINDEX

You can reprime the index and start an initial update using the SYSPROC.SYSTS_REPRIMEINDEX stored procedure. Use this stored procedure when you want to restore data from the base table.

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

XML Search Namespace Support

You can use a namespace to scope elements and attributes in a document. Namespaces are useful in restricting the query search to the meaningful elements within the document.

Overview

In XML, element and attribute names are chosen by the developer. These names can create conflicts when XML documents from different applications are mixed.

It is therefore useful to restrict the query search to the meaningful elements within the document, especially when multiple different document types might be indexed. Restricting the search can be accomplished by using namespaces.

Namespaces provide scoping of the elements and attributes of the document to ensure correct interpretation of the values. Namespaces are described with long name (URI) and optionally a short name called the Qname (qualified name).

```
<?xml version='1.0'?>
  <doc xmlns:x="http://example.com/ns/abc">
```

```
<x:p/>
</doc>
```

`http://example.com/ns/abc` is the long namespace name and `x` is the QName prefix. A QName prefix is useful as a shorthand for the namespace of each element reference.

Element `p` is qualified by namespace `http://example.com/ns/abc`.

The default namespace

A default namespace can be specified for XML elements. The default namespace applies to the current tag and any descendent tags. Any unqualified tag in the namespace inherits the default namespace.

```
<?xml version='1.0'?>
  <doc xmlns="http://example.com/ns/abc">
    <p/>
  </doc>
```

In this case both `doc` and `p` elements are in the `http://example.com/ns/abc` namespace.

Attribute namespaces

An attribute might have a different namespace than its associated element.

Element and attribute, qualified:


```
<dog xmlns:an="http://example.org/animals" xmlns:sz="http://example.org/sizes">
  <an:breed sz:size="Medium">Mutt</an:breed>
</dog>
```

There is a difference in how elements and attributes inherit namespace when it is not explicitly specified. Unqualified elements pick up the default namespace of the scope within which they lie. Unqualified attributes do not have any namespace.

Element and attribute, non-qualified:

```
<dog xmlns:an="http://example.org/animals">
  <breed size="Medium">Mutt</an:breed>
</dog>
```

In this example, element `breed` has a namespace of `http://example.org/animals`. However, attribute `size` has no namespace associated with it.

For more information about XML namespaces, consult the W3C Recommendation for Namespaces in XML which can be found at the [World Wide Web Consortium\(W3C\)](http://www.w3.org) (<http://www.w3.org>) .

Reserved QName prefixes

The following QName prefixes are reserved and must not be used to qualify user-defined elements or attributes: **xml**, **xs**, **xsi**, **fn**, **local**.

Using namespaces in search

QName prefixes and default element namespaces must be defined in the `@xmlxp` query prolog of the search term.

An example prolog that maps namespace `ns1` to URI `"http://mycompany.com"`

```
declare namespace ns1 = "http://mycompany.com";
```

An example prolog that specifies that all unqualified elements are qualified by URI `"http://mycompany.com"`:

```
declare default element namespace "http://mycompany.com"
```

If a query does not declare any namespace QName prefix or default element namespace, then namespaces are not considered in the query. An element or attribute name is considered a match if it exists in any namespace.

If any QName prefix or default namespace is declared, element or attribute names are a match only if they exist in the namespace specified.

The syntax

```
declare default element namespace "";
```

could be used to indicate that unqualified tags are not in any namespace.

QName prefixes used in the XML search string are NOT required to match the QName prefix used in the XML document. Matches are based solely on the long name URI.

Examples:

Restrict search to attribute `attr` of element `test` where element `test` is mapped to namespace `"http://posample.org"`, and `attr` is not in any namespace. Use default namespace to simplify syntax.

```
CONTAINS(myxmlcol, '@xmlxp:''declare default element namespace "http://myexample.org";  
/test[@attr > xs:date("2005-01-01")]''')
```

Restrict search to attribute `attr` of element `test` where element `test` has a namespace of `"http://myexample.org"`. Use explicit namespace syntax by using the QName prefix `abc`.

```
CONTAINS(myxmlcol, '@xmlxp:''declare namespace abc = "http://myexample.org";  
/abc:test[@attr < xs:date("2009-01-01")]''')
```

Restrict search to `shipTo` name and `billTo` name child elements of element `purchaseOrder`, which is explicitly mapped to namespace `"http://myexample.org"` using QName prefix `ns1`. A default namespace is also defined (`"http://mastsample.org"`), which applies to `shipTo`, `name`, and `billTo`.

```
CONTAINS(myxmlcol, '@xmlxp:'' declare default namespace "http://mastsample.org";  
declare namespace ns1 = "http://posample.org"; /ns1:purchaseOrder[shipTo/name = "Jane"  
and billTo/name = "Jason"]''')
```

Restrict search to attribute `name` (explicitly defined in namespace `"http://posample.org"`) of `shipTo` element (in default namespace `"http://mastsample.org"`), which is a child of element `purchaseOrder` (explicitly defined in namespace `"http://posample.org"`). The default namespace `"http://mastsample.org"` applies to elements `shipTo`, `billTo` and `name`.

```
CONTAINS(myxmlcol, '@xmlxp:'' declare default namespace "http://mastsample.org";  
declare namespace ns1 = "http://posample.org"; /ns1:purchaseOrder/shipTo[@ns1:name =  
"Jane" and billTo/name = "Jason"]''')
```

XML Search Example

- Create a table `XML_DOCUMENTS` in schema `XMLTEST` to store the XML documents:

```
CREATE TABLE XMLTEST.XML_DOCUMENTS (ID INT, XML_DATA XML, PRIMARY KEY (ID));
```

- Create a text search index called `XML_INDEX` over the XML column:

```
call SYSPROC.SYSTS_CREATE('XMLTEST', 'XML_INDEX', 'XMLTEST.XML_DOCUMENTS(XML_DATA)', '');
```

- Insert some XML Documents:

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)  
VALUES(1,
```

```
'<BOOK publication_date="2009-01-01">' ||
<TITLE> OmniFind Text Search Server for DB2 </TITLE>' ||
<ID_NUMBER> 1 ></ID_NUMBER>' ||
<CHAPTER>' ||
  <NUMBER> 1 </NUMBER>' ||
  <TITLE> Introduction </TITLE>' ||
  <ABSTRACT> This chapter will introduce the reader to the capabilities of OmniFind
  for DB2 for IBM i </ABSTRACT>' ||
</CHAPTER>' ||
<CHAPTER>' ||
  <NUMBER> 2 </NUMBER>' ||
  <TITLE> Creating a Text Search Index </TITLE>' ||
  <ABSTRACT> This chapter will explain how to create a text search index </ABSTRACT>'
||
' </CHAPTER>' ||
'</BOOK>');
```

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)
VALUES(2,
'<BOOK publication_date="2010-02-01">' ||
<TITLE> Using the XML data type for DB2 for IBM i </TITLE>' ||
<ID_NUMBER> 2 ></ID_NUMBER>' ||
<CHAPTER>' ||
  <NUMBER> 1 </NUMBER>' ||
  <TITLE> Introduction </TITLE>' ||
  <ABSTRACT> This chapter will introduce the reader to the DB2 XML data type </
ABSTRACT>' ||
  </CHAPTER>' ||
  <CHAPTER>' ||
  <NUMBER> 2 </NUMBER>' ||
  <TITLE> Inserting XML data into a DB2 table </TITLE>' ||
  <ABSTRACT> This chapter will explain how to insert XML data into a DB2 table </
ABSTRACT>' ||
  </CHAPTER>' ||
  <CHAPTER>' ||
  <NUMBER> 3 </NUMBER>' ||
  <TITLE> Searching XML data </TITLE>' ||
  <ABSTRACT> This chapter will explain how to query data in XML columns
  using the CONTAINS and SCORE UDFS </ABSTRACT>' ||
  </CHAPTER>' ||
'</BOOK>');
```

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)
VALUES(3,
'<BOOK xmlns="http://www.ibm.com/digital_media_library" ||
  publication_date="2010-02-01">' ||
<TITLE> Using Namespaces with OmniFind Text Search Server for DB2 for IBM i </
TITLE>' ||
<ID_NUMBER> 2 </ID_NUMBER>' ||
<CHAPTER>' ||
  <NUMBER> 1 </NUMBER>' ||
  <TITLE> Introduction </TITLE>' ||
  <ABSTRACT> This chapter will introduce the reader to XML namespaces </ABSTRACT>' ||
  </CHAPTER>' ||
  <CHAPTER>' ||
  <NUMBER> 2 </NUMBER>' ||
  <TITLE> Using default namespaces </TITLE>' ||
  <ABSTRACT> This chapter will explain how to use a namespace in an XML search </
ABSTRACT>' ||
  </CHAPTER>' ||
'</BOOK>');
```

- Update the index:

```
CALL SYSPROC.SYSTS_UPDATE('XMLTEST', 'XML_INDEX', '');
```

Example queries

Example 1:

Find all documents that have a root element BOOK with a direct descendant TITLE that contains DB2.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''/BOOK/TITLE[. contains("DB2")]'' ') = 1;
```

Because a namespace prolog is not specified in the search term, no namespace is considered in the search.

<i>Table 17. Result</i>	
ID	
1	
2	
3	

Example 2:

Find all documents that have a root element BOOK with a direct descendant TITLE that contains DB2. Use a default element namespace to indicate that BOOK and TITLE must be in the "http://www.ibm.com/digital_media_library" namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare default element namespace
"http://www.ibm.com/digital_media_library";
/BOOK/TITLE[. contains("DB2")]'' ') = 1;
```

<i>Table 18. Result</i>	
ID	
3	

Example 3:

Find all documents that have a root element BOOK that has an attribute publication_date after "2010-01-01" and has a child element TITLE that contains DB2. Restrict the search so that tags BOOK and TITLE must not exist in any namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare default element namespace "";
/BOOK[@publication_date > xs:date("2010-01-01")]/TITLE[. contains("DB2")]'' ') = 1;
```

<i>Table 19. Result</i>	
ID	
2	

Example 4:

Find all documents with a root element BOOK (not in any namespace) that have a direct descendant CHAPTER (also not in a namespace) that contains information about inserting data into an XML table.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare default element namespace "";
/BOOK/CHAPTER[. contains("inserting XML data into a table")]'' ') = 1;
```

Note:

- The text contained within CHAPTER includes the text contained within the ABSTRACT and TITLE elements that are the descendants of CHAPTER.
- The search string is not case-sensitive, and linguistic variations of the search words are considered.

<i>Table 20. Result</i>	
ID	
2	

Example 5:

Find all documents with a root element BOOK (in namespace "http://www.ibm.com/digital_media_library") that have a direct descendant CHAPTER (also in namespace "http://www.ibm.com/digital_media_library"). CHAPTER must have a direct descendant NUMBER (in namespace "http://www.ibm.com/digital_media_library") with a value of 1, and also contain text information about searching an XML namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare namespace ns1 = "http://www.ibm.com/digital_media_library";
/ns1:BOOK/ns1:CHAPTER[. contains("search XML using a namespace") and NUMBER = 1]'' ') = 1;
```

Document #3 is the only document with tags in the correct namespace, but it has key word matches only in a chapter with a number value of 2 (not 1).

No rows are returned.

<i>Table 21. Result</i>	
ID	

Example 6:

Find all documents with a root element BOOK(in namespace "http://www.ibm.com/digital_media_library") that have a direct descendant CHAPTER (in namespace "http://www.ibm.com/digital_media_library"). CHAPTER must have a direct descendant NUMBER (in namespace "http://www.ibm.com/digital_media_library") with a value of 1. BOOK must have a descendant CHAPTER (not necessarily with a NUMBER descendant) that contains text information about searching an XML namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare namespace ns1 = "http://www.ibm.com/digital_media_library";
/ns1:BOOK[ns1:CHAPTER contains("search XML using a namespace")]/ns1:CHAPTER[ns1:NUMBER = 1]'' ') = 1;
```

Document 3 does have a CHAPTER element that matches the CONTAINS criteria, and also has a CHAPTER element with a descendant NUMBER that has a value of 1. Therefore, document 3 is a match for this query.

<i>Table 22. Result</i>	
ID	
3	

Related reference

[SYSPROC.SYSTS_CREATE](#)

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

[SYSPROC.SYSTS_UPDATE](#)

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

[CONTAINS](#)

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

XML search example

This example includes a table of XML documents, a text search index over an XML column in the table, and six SQL text searches using CONTAINS.

Note: By using the code examples, you agree to the terms of the [“Code license and disclaimer information”](#) on page 147.

- Create a table XML_DOCUMENTS in schema XMLTEST to store the XML documents:

```
CREATE TABLE XMLTEST.XML_DOCUMENTS (ID INT, XML_DATA XML, PRIMARY KEY (ID));
```

- Create a text search index called XML_INDEX over the XML column:

```
call SYSPROC.SYSTS_CREATE('XMLTEST', 'XML_INDEX', 'XMLTEST.XML_DOCUMENTS(XML_DATA)', '');
```

- Insert some XML documents:

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)
VALUES(1,
'

```

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)
VALUES(2,
'

```

```
INSERT INTO XMLTEST.XML_DOCUMENTS (ID, XML_DATA)
VALUES(3,
'

```

```

' <TITLE> Introduction </TITLE>' ||
' <ABSTRACT> This chapter will introduce the reader to XML namespaces </ABSTRACT>' ||
' </CHAPTER>' ||
' <CHAPTER>' ||
' <NUMBER> 2 </NUMBER>' ||
' <TITLE> Using default namespaces </TITLE>' ||
' <ABSTRACT> This chapter will explain how to use a namespace in an XML search </
ABSTRACT>' ||
' </CHAPTER>' ||
' </BOOK>');

```

- Update the index:

```
CALL SYSPROC.SYSTS_UPDATE('XMLTEST', 'XML_INDEX', '');
```

Example queries

Search 1:

Find all documents that have a root element BOOK with a direct descendant TITLE that contains DB2.

```

SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlxp:''/BOOK/TITLE[. contains("DB2")]'' ') = 1;

```

Because a namespace prolog is not specified in the search term, no namespace is considered in the search.

<i>Table 23. Result</i>	
ID	
1	
2	
3	

Search 2:

Find all documents that have a root element BOOK with a direct descendant TITLE that contains DB2. Use a default element namespace to indicate that BOOK and TITLE must be in the "http://www.ibm.com/digital_media_library" namespace.

```

SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlxp:''declare default element namespace
"http://www.ibm.com/digital_media_library";
/BOOK/TITLE[. contains("DB2")]'' ') = 1;

```

<i>Table 24. Result</i>	
ID	
3	

Search 3:

Find all documents that have a root element BOOK that has an attribute publication_date after "2010-01-01" and has a child element TITLE that contains DB2. Restrict the search so that tags BOOK and TITLE must not exist in any namespace.

```

SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlxp:''declare default element namespace "";
/BOOK[@publication_date > xs:date("2010-01-01")]/TITLE[. contains("DB2")]'' ') = 1;

```

<i>Table 25. Result</i>	
ID	
	2

Search 4:

Find all documents with a root element BOOK (not in any namespace) that have a direct descendant CHAPTER (also not in a namespace) that contains information about inserting data into an XML table.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare default element namespace "";
/BOOK/CHAPTER[. contains("inserting XML data into a table")]'' ') = 1;
```

Note:

- The text contained within CHAPTER includes the text contained within the ABSTRACT and TITLE elements that are the descendants of CHAPTER.
- The search string is not case-sensitive, and linguistic variations of the search words are considered.

<i>Table 26. Result</i>	
ID	
	2

Search 5:

Find all documents with a root element BOOK (in namespace "http://www.ibm.com/digital_media_library") that have a direct descendant CHAPTER (also in namespace "http://www.ibm.com/digital_media_library"). CHAPTER must have a direct descendant NUMBER (in namespace "http://www.ibm.com/digital_media_library") with a value of 1, and also contain text information about searching an XML namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare namespace ns1 = "http://www.ibm.com/
digital_media_library";
/ns1:BOOK/ns1:CHAPTER[. contains("search XML using a namespace") and NUMBER = 1]'' ') = 1;
```

Document #3 is the only document with tags in the correct namespace, but it has key word matches only in a chapter with a number value of 2 (not 1).

No rows are returned.

<i>Table 27. Result</i>	
ID	

Search 6:

Find all documents with a root element BOOK(in namespace "http://www.ibm.com/digital_media_library") that have a direct descendant CHAPTER (in namespace "http://www.ibm.com/digital_media_library"). CHAPTER must have a direct descendant NUMBER (in namespace "http://www.ibm.com/digital_media_library") with a value of 1. BOOK must have a descendant CHAPTER (not necessarily with a NUMBER descendant) that contains text information about searching an XML namespace.

```
SELECT ID
FROM XMLTEST.XML_DOCUMENTS
WHERE CONTAINS(XML_DATA, '@xmlns:''declare namespace ns1 = "http://www.ibm.com/
digital_media_library";
```

```
/ns1:BOOK[ns1:CHAPTER contains("search XML using a namespace")]/ns1:CHAPTER[ns1:NUMBER = 1]''
') = 1;
```

Document 3 does have a CHAPTER element that matches the CONTAINS criteria, and also has a CHAPTER element with a descendant NUMBER that has a value of 1. Therefore, document 3 is a match for this query.

<i>Table 28. Result</i>	
ID	
3	

XML search query grammar

The grammar for XML Search is based on a subset of the XPath language, which is defined by Extended Backus-Naur Form (EBNF) grammar. Queries that do not conform to the supported grammar are rejected by the query parser.

The EBNF grammar has been simplified in the following ways by:

- Disallowing absolute path names in predicate expressions.
- Recognizing only one axis (tag) and only in the forward direction.
- Applying additional semantic restrictions to the use of the Wildcard character (see previous section on "Path Expression Wildcard Support" in "XML search" on page 49).
- Requiring that the namespace declaration is specified in the search string before any usage, implied or explicit, of the namespace. If the namespace declaration is not included, namespaces are not considered in the search.
- Relative path expressions must have a tag or attribute name included in the expression. The query '/' to select the root node, and '//'' to select all nodes are not valid expressions.

The following table shows the supported grammar in EBNF notation.

<i>Table 29. Supported query grammar in EBNF notation</i>	
Symbol	Production
XMLQuery ::=	QueryPrefix NameSpaceDeclaration QueryString QueryPrefix QueryString
QueryPrefix ::=	@xmlxp:
QueryString ::=	"" PathExpr ""
PathExpr ::=	RelativePathExpr "/" RelativePathExpr? "//" RelativePathExpr
RelativePathExpr ::=	StepExpr (("/" "//") StepExpr)*
StepExpr ::=	("." AbbrevForwardStep) Predicate?
AbbrevForwardStep ::=	"@"? (QName "*")
Predicate ::=	"[" PredicateExpr "]"
PredicateExpr ::=	Expr PredicateExpr ("and" "or") "(" PredicateExpr ")"
Expr ::=	ComparisonExpr ContainmentExpr
ComparisonExpr ::=	PathExpr ComparisonOp Literal

Table 29. Supported query grammar in EBNF notation (continued)	
ComparisonOp ::=	"=" "<" ">" "!=" "<=" ">="
Literal ::=	StringLiteral NumericLiteral DateLiteral
ContainmentExpr ::=	PathExpr "contains" "(" StringLiteral ")" PathExpr "excludes" "(" StringLiteral ")"
StringLiteral ::=	"\" [^]* \"\ \"\" [^']* \"\"
DateLiteral ::=	"xs:date(\"\" xmlDate \"\")" "xs:dateTime(\"\" xmlDateTime \"\")"
xmlDate ::=	yyyy"- "mm"- "dd
xmlDateTime ::=	yyyy"- "mm"- "dd [T] hh": "mm": "ss". "uuuuuu
NamespaceDeclaration ::=	defaultNamespace (NamespacePrefixDeclaration)*
defaultNamespace ::=	"declare default element namespace " StringLiteral ";"
NamespacePrefixDeclaration ::=	"declare namespace" NamespacePrefix "=" StringLiteral ";"
NamespacePrefix ::=	[^":]+

Administer an OmniFind(r) Text Search Server for DB2(r) for i

You can administer the OmniFind Text Search Server for DB2 for i using the following techniques and tools.

Start the OmniFind Text Search Server for DB2 for i

You can start the OmniFind Text Search Server for DB2 for i by calling `SYSPROC.SYSTS_START`.

OMNIFIND starts the text search server automatically as needed, as long as the `SERVERSTATUS` in `QSYS2.SYSTEXTSERVERS` is 0. This policy allows the text search server to start automatically when the host system starts. However, you can start the server manually if necessary.

To start the server:

```
CALL SYSPROC.SYSTS_START(serverid)
```

If successful, the `SERVERSTATUS` in `QSYS2.SYSTEXTSERVERS` is set to 0 after you call the procedure. When the server is local, the following jobs are active in the background:

- `QJVAEXEC QDBTS BCI 0.0 JVM-com.ibm.es`
- `QJVAEXEC QDBTS BCI 0.0 PGM-textExtrac`
- `QJVAEXEC QDBTS BCI 0.0 PGM-textExtrac`
- `QJVAEXEC QDBTS BCI 0.0 PGM-textExtrac`
- `QJVAEXEC QDBTS BCI 0.0 PGM-textExtrac`

where `QDBTS` is the OmniFind user profile created while installing product.

It might take a few minutes before all these jobs are active and the text server can be used.

Related reference

SYSPROC.SYSTS_START

You can enable DB2 text search functions by calling the SYSPROC.SYSTS_START stored procedure.

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

Stop the OmniFind Text Search Server for DB2 for i

You can stop the OmniFind Text Search Server for DB2 for i manually by using the shutdown script that is provided.

If you installed the OmniFind Text Search Server for DB2 for i as a service, the text search server stops automatically each time that the host system is shut down. However, you can stop the server manually even if you installed the OmniFind Text Search Server for DB2 for i as a service.

To stop the OmniFind Text Search Server for DB2 for i:

1. Indicate in the SYSTEXTSERVER catalog that the server is stopped by calling SYSPROC.SYSTS_STOP.

- To stop all servers: CALL SYSPROC.SYSTS_STOP().
- To stop a specific server:

- a. Query the server catalog to get the *serverid* that you want to stop:

```
SELECT SERVERID, SERVERPORT, SERVERSTATUS, SERVERPATH
FROM QSYS2.SYSTEXTSERVERS
```

Note: SERVERPATH identifies the server. SERVERSTATUS indicates whether the server is currently active (0) or inactive (1).

- b. Call SYSPROC.SYSTS_STOP, specifying the numeric *serverid* or the alias name of the server you want to stop:

```
CALL SYSPROC.SYSTS_STOP(serverid).
```

2. Indicate in the SYSTEXTSERVER catalog that the server is stopped by calling SYSPROC.SYSTS_SHUTDOWN. Beside set SERVERSTATUS to inactive(1), this procedure will also end the text search server jobs on the host system.

- To shutdown all servers: CALL SYSPROC.SYSTS_SHUTDOWN().
- To shutdown a specific server:

- a. Query the server catalog to get the *serverid* that you want to shutdown:

```
SELECT SERVERID, SERVERPORT, SERVERSTATUS, SERVERPATH
FROM QSYS2.SYSTEXTSERVERS
```

Note: SERVERPATH identifies the server. SERVERSTATUS indicates whether the server is currently active (0) or inactive (1).

- b. Call SYSPROC.SYSTS_SHUTDOWN, specifying the numeric *serverid* or the alias name of the server you want to shutdown:

```
CALL SYSPROC.SYSTS_SHUTDOWN(serverid).
```

3. (Optional) Stop the server itself by calling the shutdown script. Stopping the server ends all the text search server jobs on the host system.

Stop the server in the Qshell environment.

To shut down the local server, enter the following command from the command line:

```
QSH CMD('cd /QOpenSys/QIBM/ProdData/TextSearch/server1/bin; shutdown.sh')
```

If the server to be shut down is not the default local server created by the install process, you need to obtain the correct SERVERPATH value from QSYS2.SYSTEXTSERVERS. Use that SERVERPATH instead of /QOpenSys/QIBM/ProdData/TextSearch/server1/bin.

If you stop the server by using the shutdown script, the SERVERSTATUS catalog is not changed to the inactive (1) status. When the SYSTS_CREATE, SYSTS_UPDATE, and SYSTS_DROP stored procedures are called the next time, or when a CONTAINS or SCORE built-in function is invoked as part of an SQL query, the server will start automatically.

Related reference

SYSPROC.SYSTS_STOP

You can call the SYSPROC.SYSTS_STOP stored procedure to stop DB2 text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped).

SYSPROC.SYSTS_SHUTDOWN

You can call the SYSPROC.SYSTS_SHUTDOWN stored procedure to shutdown DB2® text search functions. This stored procedure sets the SERVERSTATUS value in the catalog QSYS2.SYSTEXTSERVERS to 1 (stopped) and also ends the text search server jobs on the host system.

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

Save and restore text search indexes

You can save and restore your text search indexes with or without data.

Save and restore a text search index without data

You can save and restore a text search index structure without the index data. The save and restore process can be accomplished using the SAVOBJ and SAVLIB CL commands.

When you create a text search index using SYSTS_CREATE, a DB2 view is created using the index schema and name as the view name. The view serves as a mechanism for saving and restoring the structure of the index.

The user can save the view using the same methods for saving database tables and views. (See SAVOBJ or SAVLIB CL commands.) Saving the view automatically saves additional information needed to recreate the index during restore.

The view can be restored using the RSTOBJ or RSTLIB CL command. DB2 for i recognizes that the view represents a text search index and recreates the index. After the index structure has been recreated, an update will be submitted to a background job to repopulate the index data.

Additional considerations need to be made during the restore process:

1. If the text search server cannot be started, or a required product is not installed on the system, the restore fails. See the [Software requirements](#) for a list of required products.
2. If the text search index exists on the system, the following actions are taken.
 - a. If the existing index information exactly matches the index being restored, the restore succeeds. The index is not rebuilt.
 - b. If the existing index information does not match the index being restored, and cannot be modified to match without recreating the index, the restore fails.
 - c. If the existing index information does not match the index being restored, but can be modified to match using SYSTS_ALTER, then the existing index is altered to match the index that was saved. The index is not rebuilt.
3. The index is restored to use the same text search server that was in use at the time of the save. If the server that was used at the time of the save is not defined, a currently available server is selected. If the saved server is defined but not available, the restore fails.

4. If the text search index cannot be created for any other reason, such as an incompatible column in the based-on table, the restore fails.
5. The staging table name in QSYS2, trigger names that are added to the based-on table, and the collection name on the text search server can change, since they are generated by the system.
6. Synonyms that have been added to the text search index synonym dictionary are not preserved.
7. If the index exists in the System catalogs at restore time, and the view does not currently exist on the system, only the view is restored. The staging table, text search server collection, and triggers on the based-on table are not created.

In this case, the text search index is assumed to be part of a larger restore where the individual pieces of the index were saved explicitly by the user, and are now all being restored (such as restoring the entire system).

All the required pieces of the index must be restored before the index works. It is the users responsibility to ensure that all pieces of the index are synchronized.

Text search indexes are supported by the Restore Deferred Objects (RSTDFROBJ) command. The use of the DFRID parameter on the RSTOBJ and RSTLIB CL commands is recommended. This parameter allows Text Search indexes to be restored using the RSTDFROBJ command after correcting common conditions that prevent creation of the index.

Examples of conditions that cause the index to be deferred are:

- A required product was not installed.
- A text search server was not available or defined.
- The based on table did not exist.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

SYSPROC.SYSTS_ALTER

You can call the SYSPROC.SYSTS_ALTER stored procedure to modify attributes of an index that was created by SYSPROC.SYSTS_CREATE. Only attributes explicitly specified on this procedure are changed. All other attributes of the index remain unchanged.

Related information

Save Object (SAVOBJ)

Save Library (SAVLIB)

Restore Object (RSTOBJ)

Restore Library (RSTLIB)

Restore Deferred Objects (RSTDFROBJ)

Save and restore a text search index with data

Saving and restoring a text search index with data is a more completed operation than without data.

You must save the following objects:

- The text search index (stored in the integrated file system).
- The staging table used as a log file that tracks record changes in the base table (over which the index is built). The staging table is in library QSYS2. Its name begins with QDBTS, for example, QDBTS00001.
- The view, which is the database object representing the text index. The view has the same name as the text index.
- The base table over which the index is built.
- The SQL catalogs that store the information to track the index.

Complete the following steps to save the text search indexes:

1. Recommended: bring the indexes up to date by first performing update operations (SYSTS_UPDATE) for the text search indexes.
2. Save the base table and view using standard save techniques such as the SAVOBJ command.
3. Save the staging tables that are in QSYS2 using standard save techniques. For example, SAVOBJ LIB(QSYS2) OBJ(QDBTS*).
4. Save the text search index catalogs in QSYS2:

The catalog names all begin with SYSTXT, for example, SYSTXTSRVR. Like the other SQL catalogs in QSYS2, it is your responsibility to ensure that a backup copy is saved and available.

This backup copy can be accomplished in one of two ways:

- a. The entire library can be saved as part of the SAVLIB command, specifying parameter LIB as either *ALLUSR or *IBM.
 - b. The specific text search catalogs can be saved using the SAVOBJ command, specifying LIB(QSYS2) and OBJ((SYSTXT*)).
5. Save the text search index information in the integrated file system. This information includes the entire contents of the *config* directory under the text server path.

The text server path can be determined by querying the SERVERPATH column of the SYSTXTSRVR catalog for the server of interest. The server path has the directory bin appended to it, which you replace with the config directory.

A common save technique is to use the SAV command, though any type of save compression works.

Note: This save information is only applicable to text servers running on IBM i.

Example:

Suppose you want to save all the text indexes associated with the default text server created by OMNIFIND. You have a table QGPL/MYDOCS with text index QGPL/MYDOCIX built over it. In this example, the save media are save files.

Complete the following steps:

1. Save all the staging tables and the OMNIFIND catalogs from QSYS2:

```
SAVOBJ OBJ(QDBTS* SYSTXT*) LIB(QSYS2) DEV(*SAVF) SAVF(QGPL/SAVFQSYS2)
```

2. Save the base table and view:

```
SAVOBJ OBJ(MYDOCS MYDOCIX) LIB(QGPL) DEV(*SAVF) SAVF(QGPL/SAVFMYFILE)
```

3. Using SQL, get the path name of the text server. In this example, serverid = 2:

```
SELECT SERVERPATH FROM systxtsrvr WHERE serverid=2
```

The SERVERPATH value returned is /QOpenSys/QIBM/ProdData/TextSearch/server1/bin/.

Note: Verify that you are querying for the correct server.

4. Substitute config for bin/ and save the text indexes:

```
SAV DEV('/QSYS.LIB/QGPL.LIB/SAVIFS.FILE') OBJ((' /QOpenSys/QIBM/ProdData/  
TextSearch/server1/config'))
```

The text indexes are now saved in save files QGPL/SAVFMYFILE, QGPL/SAVFQSYS2, and QGPL/SAVIFS.

The text index restore must be done in the same order as the save. The QSYS2 catalogs MUST be restored first.

Related reference

[SYSPROC.SYSTS_UPDATE](#)

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

Related information

[Save Object \(SAVOBJ\)](#)

[Save Library \(SAVLIB\)](#)

Problem determination

You can use the system and trace messages logged to determine the source of problems that might occur.

The OmniFind Text Search Server for DB2 for i server logs are located in the <INSTALL_HOME>/log directory. The default server log created at installation is located in the /QOpenSys/QIBM/ProdData/TextSearch/server1/log directory.

By default, the trace log is turned off, and the system log level is set to `informational`. You can use the configuration tool to change the trace and log level options.

You can view the server logs directly in the log directory.

View server logs

The server logs are located in integrated-file-system directory /QOpenSys/QIBM/ProdData/TextSearch/server<servername>/log. These logs can help you determine the source of problems.

The following log files are written to the log directory:

admin_audit*.csv

Collects information about configuration changes that are made by the API.

commandLineTools*.log

Records command-line tool information, warning, and error messages.

default*.log

Contains information, warning, and error messages that are logged during server activity.

documentErrors*.log

Contains warning and error messages about documents that were not indexed because their file formats are not supported. These document warning and error messages are not recorded in the trace and default logs.

monitor*.csv

Contains information about the indexing status, queue status, query load, and other system statistics.

serverConfiguration.log

Provides information about the server configuration on startup.

trace*.log

Contains trace information, warning, and error messages. You can enable tracing by using the administration tool and set the log level by using the configuration tool.

version.log

Records the history of version changes for some of the main components of the server, such as the server JAR file.

Administration tools

OMNIFIND provides tools that you can use for common tasks. These tasks include configuring and administering an additional text search server, and adding a synonym dictionary to a collection.

These tools are shell scripts rather than CL commands. They can be called within the script environment that is started through either the Start QSH (STRQSH) or QSH (QSH) CL commands.

These tools do not authenticate user IDs. However, these tools can be run only by a user with valid access to the text search server.

Related information

[Start QSH \(STRQSH\)](#)

[Start QSH \(QSH\)](#)

Configuration tool

Use the configuration tool to customize configuration settings after you install OmniFind Text Search Server for DB2 for i.

To customize most of the configuration settings, you must stop the text search server before running the configuration tool.

However, when the server is running, you can display the following options:

- the current authentication token
- the server port
- the current properties of the system

The configServerAndDB2 tool

The configServerAndDB2 (configServerAndDB2.sh) tool is located in integrated-file-system directory /QOpenSys/QIBM/ProdData/TextSearch. This tool can be used to create or modify entries in the DB2 catalog file SYSTEXTSERVERS.

It can also be used to configure the authentication token or the port number associated with the specific server. The tool modifies or sets the values for SERVERAUTHTOKEN and SERVERPORT in the DB2 catalog file SYSTEXTSERVERS.

If you want to create an additional server that runs locally to your system, use the [“ServerInstance tool”](#) on page 85 instead.

The configServerAndDB2 (configServerAndDB2.sh) tool is called with five parameters:

1. The first parameter is either **generateToken** or **configureHTTPListener**.
2. The second parameter is **-serverPath**.
3. The third parameter is the path to the root node in the integrated file system where the information related to the server is stored. Example: /QOpenSys/QIBM/ProdData/TextSearch/server2.
4. The fourth and fifth parameters vary depending on the value of the first parameter.
 - If the first parameter is **generateToken**, then the fourth parameter is **-seed** followed by an integer (for example, 1) as the fifth parameter
 - If the first parameter is **configureHTTPListener**, then the fourth parameter is **-adminHTTPPort**. The fifth parameter is an integer value that is used as the socket port for the server.

Here are two examples:

- STRQSH
cd /QOpenSys/QIBM/ProdData/TextSearch
configServerAndDB2.sh generateToken -serverPath /QOpenSys/QIBM/ProdData/TextSearch/server2
-seed 1
- STRQSH
cd /QOpenSys/QIBM/ProdData/TextSearch
configServerAndDB2.sh configureHTTPListener -serverPath /QOpenSys/QIBM/ProdData/TextSearch/
server2
-adminHTTPPort 9997

The configTool script

The configTool.sh script is available for each local server. It is located in Integrated File System (IFS) directory:

/QOpenSys/QIBM/ProdData/TextSearch/server<servername>/bin.

This tool can be used to list and set parameters for OmniFind Text Search Server. Prior to using it to modify server configuration, it is recommended that you review the following OmniFind article to understand how to tune OmniFind:

- [Introduction to IBM i OmniFind \(ECM Text Search component\) upgrade](#)

Restriction: Some parameters are read only, while others can only be modified when the OmniFind Text Search Server is stopped.

Authorization

To use the configTool.sh tool, the user must have the following privilege:

- *ALLOBJ special authority

Syntax

configTool.sh <command> [-command_options] [-locale value] [-configPath value]

Commands and command options

list -system [-details] [-showAdvanced][-parameter_name]

Prints system-level parameters and their current values. See the following sections for descriptions of each parameter.

-details

Prints detailed information about each parameter, including a description, default value, type, and whether it is modifiable.

-showAdvanced

Prints information about advanced configuration parameters.

-parameter_name

Prints detailed information about the specified parameter.

set -system -parameter_name value

Specifies the value of a system-level parameter.

generateToken

Generates the authentication token.

printToken

Prints the current authentication token and encryption key.

sysinfo

Prints system information, such as the build version number, operating system, and JAR manifest version.

Parameters

-configPath

Specifies the absolute path to the configuration directory. This directory contains configuration files and the \collections subdirectory. This parameter cannot be modified.

-directoryTypeForSearch

Advanced: The type of file system to use for search. Specify one of the following values:

mmap Specify mmap to store shared indexes by using memory mapped files. Using mmap improves search performance, especially for concurrent searches. Using this option results in increased heap memory consumption.

simple Specify simple for standard file input/output.

noifs Specify noifs to store shared indexes by using New Input/Output (default for OmniFind Text Search Server for DB2 for i).

Important: When you specify **mmap**, it is recommended to configure 64-bit JVM for OmniFind Text Search Server and increase the heap size accordingly by specifying the **maxHeapSize** parameter to avoid out-of-memory error.

-logFolder

Specifies the absolute path to the log directory.

-maxDocumentSizeBytes

Specifies the maximum size (in bytes) of documents that are indexed. The default value is 120000000.

-maxHeapSize

Specifies the maximum heap size of the server. Set this value according to JVM heap size specifications. For example, the maximum heap size that can be allocated to a 32-bit Java virtual machine is 1.8 GB. The default value is 1500M for 32-bit JVMs and 4G for 64-bit JVMs (M=megabytes, G=gigabytes). The value must be an integer. For example, to specify 1.8 gigabytes, enter 1800M. This parameter can be modified only when the server is stopped.

Attention: In OmniFind Text Search Server ECMTS version 1.5.0, **maxHeapSize** was called **startupHeapSize**.

-numberOfIndexerThreads

Advanced: Specifies the number of indexing threads that run on the server. The default value is 8.

-numberOfTokenizers

Specifies the number of language processors that are used for parsing input into tokens. The default value is 20.

-port

Specifies the number of the port on which the OmniFind Text Search server listens for requests.

-queryCacheSizeMB

Advanced: Specifies the size (in MB) of the query cache. This parameter can be modified only when the server is stopped. The default value is 200 MB.

-searchableRefreshRateMS

Advanced: Time period (in milliseconds) for checking whether the index has changed and the searchable must be reopened. The default value is 1.

-tempDirectory

Specifies the absolute path to the temporary directory.

-useQueryCache

Advanced: Enables query caching. The default value is not to use the query cache (false).

Important: when you enable the query cache, it is recommended to configure 64-bit JVM for OmniFind Text Search Server and increase the heap size accordingly by specifying the **maxHeapSize** parameter to avoid out-of-memory error.

Global arguments

-configPath

Specifies the absolute path to the configuration folder that contains the config.xml file. This global argument is mandatory.

-locale

Optional: Specifies the two- or five-character locale setting for writing messages to the trace file. If you do not specify this setting, the default value, en (English), is used.

<i>Table 30. The supported locales</i>	
Locale value	Language
cs	Czech
da	Danish
de	German
en	English
es	Spanish
fi	Finnish
fr	French
hu	Hungarian
it	Italian
ja	Japanese
ko	Korean
nl	Dutch
no	Norwegian
pl	Polish
pt	Portuguese
pt_BR	Brazilian Portuguese
ru	Russian
sv	Swedish
zh	Chinese
zh_TW	Chinese-Taiwanese

Replaced commands and arguments

With IBM i 7.3, the IBM i OmniFind Text Search Server was upgraded from ECM Text Search (ECMTS) version 1.5.0 to 5.2.1. By upgrading the ECMTS version, DB2 for i OmniFind users received infrastructure improvements, security fixes, and serviceability enhancements. These changes also result in the replacement of some OmniFind configuration commands and arguments. Review the list below to understand the configuration tool changes.

configureParams

Use the ***set -system*** command instead.

-adminHTTPPort

Use the ***-port*** parameter instead.

-logPath

Use the ***-logFolder*** parameter instead.

-tempDirPath

Use the ***-tempDirectory*** parameter instead.

-numberOfIndexers

Use the ***-numberOfIndexerThreads*** parameter instead.

-maxDocumentSize

Use the ***-maxDocumentSizeBytes*** parameter instead.

printAll

Use the ***list -system*** command instead.

printAdminHTTPPort

Use the ***list -system -port*** command instead.

Examples

Assume the configuration directory is `/QOpenSys/QIBM/ProdData/TextSearch/server1/config`.

- Print the current authentication token.

```
configTool.sh printToken -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

- List the maximum of documents size (in bytes) that can be indexed.

```
configTool.sh list -system -details -maxDocumentSizeBytes -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

- Set the maximum heap size of the server to 1800M.

```
configTool.sh set -system -maxHeapSize 1800M -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

- Increase the index change checking time period to one minute, to reduce index checking contention with search performance.

```
configTool.sh set -system -searchableRefreshRateMS 60000 -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

- Use mmap to improve search performance.

```
configTool.sh set -system -directoryTypeForSearch mmap -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

- Enable the query cache to improve search performance and reduce overall CPU consumption.

```
configTool.sh set -system -useQueryCache true -configPath /QOpenSys/QIBM/ProdData/TextSearch/server1/config
```

Related information

[Start QSH \(QSH\)](#)

SYSPROC.SYSTS_REMOVE

You can remove orphaned indexes with the SYSPROC.SYSTS_REMOVE SQL stored procedure.

Authorization

The collection-name of the possible orphaned indexes can be identified by using the QDBTS_LISTINXSTS User Defined Table Function (UDTF).

The privileges held by the authorization ID of the statement must include at least one of these privileges:

- *JOBCTL authority
- QIBM_DB_SQLADM security special function usage

Syntax

```
>>-SYSPROC.SYSTS_REMOVE ( collection-name ) -><
```


Parameter

collection-name

Specifies a string literal that identifies the name of the collection to be removed.

Note: This procedure uses the `adminTool.sh` shell script to remove the collection directory. To use this shell script, the server must be in the working state. If the server is not started, this procedure returns an error message.

SQL for SYSTS_REMOVE

```
CREATE PROCEDURE SYSPROC.SYSTS_REMOVE(  
    IN COLLECTIONNAME VARCHAR(255) CCSID 1208)  
    EXTERNAL NAME QDBTSLIB.DSN5RMCOLL  
    DYNAMIC RESULT SETS 0  
    LANGUAGE C++  
    PARAMETER STYLE SQL  
    PROGRAM TYPE MAIN  
    COMMIT ON RETURN NO  
    INHERIT SPECIAL REGISTERS;
```

Examples

- To remove an orphaned index with a collection name of `0_65_2815_2008_06_02_11_58_22_901726` from the ASP group `*SYSBASE`, enter the following command from any SQL interface:

```
CALL SYSPROC.SYSTS_REMOVE('0_65_2815_2008_06_02_11_58_22_901726')
```

The `SYSTS_REMOVE` stored procedure checks whether the index information is in catalog table `QSYS2.SYSTEXTINDEXES`. If it is true, error message `DSX_INDEX_EXIST` is returned; if not, the procedure searches under the `config/collections` directory of server 65.

If the collection does not exist, error message `DSX_COLLECTION_NOT_FOUND` is returned; if the collection exists, the procedure calls `adminTool.sh` to remove the collection.

Then the procedure checks the directory again to see whether the collection has been removed. If the collection is not removed, error message `DSX_REMOVE_COLLECTION_FAILED` is returned to the user.

Note: When the collection on the text search server is in an independent ASP group, the thread that calls the `SYSTS_REMOVE` stored procedure must run in the namespace of the independent ASP. Use the [Set Auxiliary Storage Pool Group \(SETASPGRP\)](#) command.

- To remove an orphaned index with a collection name of `33_7_26_2008_06_18_21_28_39_407824` from an independent ASP `iaspXXX`, you can use the following commands:

CL:

```
SETASPGRP(iaspXXX)
```

SQL:

```
CALL SYSPROC.SYSTS_REMOVE('33_7_26_2008_06_18_21_28_39_407824')
```

Note: If you use System i® Navigator, right-click the database name for the independent ASP, and run your SQL scripts.

Related reference

[QSYS2.SYSTEXTINDEXES administration table](#)

You can see information about each text search index in the `QSYS2.SYSTEXTINDEXES` administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

[Find orphaned and missing indexes](#)

You can find orphaned and missing indexes using an SQL User Defined Table Function (UDTF) named QDBTS_LISTINXSTS.

SYSPROC.SYSTS_REPRIMEINDEX

You can reprime the index and start an initial update using the SYSPROC.SYSTS_REPRIMEINDEX stored procedure. Use this stored procedure when you want to restore data from the base table.

If the data from the base table is restored, the updated content of the base table cannot be indexed while the SYSTS_UPDATE stored procedure is called. In this case, the SYSPROC.SYSTS_REPRIMEINDEX stored procedure can be called to reprime the index.

Note: If a synonym dictionary has been created for the text search index, this process removes the dictionary.

Syntax

```
>>-SYSPROC.SYSTS_REPRIMEINDEX( indexSchema, indexName, options) -><
```

The schema qualifier is SYSPROC.

Parameters

indexSchema

Identifies the schema of the text search index. If this parameter is null, the value of the CURRENT SCHEMA special register for the invoker is used.

The data type of this parameter is VARCHAR(128).

indexName

Identifies the name of the text search index. The name of the text search index with the index schema uniquely identifies the full-text index in the DB2 subsystem. You must specify a value that is not null for this parameter.

The data type for this parameter is VARCHAR(128).

options

A character string that specifies options that can be added in the future for this stored procedure.

Important: You must specify a null value for the *options* parameter. Otherwise, errors can be generated. Read the following Example for how to specify the *options* parameter.

SQL for SYSTS_REPRIMEINDEX

```
CREATE PROCEDURE SYSPROC.SYSTS_REPRIMEINDEX(  
    IN INDEXSCHEMA VARCHAR(128) CCSID 1208,  
    IN INDEXNAME VARCHAR(128) CCSID 1208,  
    IN OPTIONS VARCHAR(32000) CCSID 1208)  
EXTERNAL NAME QDBTSLIB.DSN5RPMIDX  
DYNAMIC RESULT SETS 0  
LANGUAGE C  
PARAMETER STYLE SQL  
MODIFIES SQL DATA  
PROGRAM TYPE MAIN  
COMMIT ON RETURN NO  
INHERIT SPECIAL REGISTERS
```

Example

- To reprime an index from any SQL interface, type the following command from any SQL interface:

```
CALL SYSPROC.SYSTS_REPRIMEINDEX('indexSchema1','indexName1','')
```

Related reference

[SYSPROC.SYSTS_UPDATE](#)

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

SYSTS_CLEAR_INDEXES

You can remove orphaned indexes with the SYSPROC. SYSTS_CLEAR_INDEXES SQL stored procedure. Another implicit way is invoking SYSTS_START directly, which tries to clear orphaned indexes automatically.

Authorization

The orphaned indexes can be identified by using the QDBTS_LISTINXSTS User Defined Table Function (UDTF).

The privileges held by the authorization ID of the statement must include at least one of these privileges:

- *JOBCTL authority
- QIBM_DB_SQLADM security special function usage

Syntax

```
>>- SYSTS_CLEAR_INDEXES
--(-+-----+--)------><
  +-serverid--+
  '-aliasname-'
```

Parameters

serverid or aliasname

Specifies the identifier of the server for clear orphaned indexes. A *serverid* or server *aliasname* is a string. If no identifier is provided, the default is to clear orphaned indexes on all servers. The identifier string must either be a valid *serverid* that exists in the SERVERID column, or a valid server aliasname that exists in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS table. If the identifier can be converted to an integer value, it is interpreted as a *serverid*. If the identifier cannot be converted to an integer value, it is interpreted as a server *aliasname*.

The data type of this parameter is VARCHAR(128).

Example

- Clear all the orphaned indexes:

```
Call SYSPROC.SYSTS_CLEAR_INDEXES();
```

- Clear orphaned indexes on a specific server with ID 50:

```
Call SYSPROC.SYSTS_CLEAR_INDEXES(50);
Call SYSPROC.SYSTS_CLEAR_INDEXES('50');
```

- Clear orphaned indexes on a specific server with alias name “Local_server”:

```
Call SYSPROC.SYSTS_CLEAR_INDEXES('Local_server');
```

- Implicitly clear orphaned indexes:

```
CALL SYSPROC.SYSTS_START();
CALL SYSPROC.SYSTS_START(50);
```

Note: When the collection on the text search server is in an independent ASP group, the thread that calls the SYSTS_CLEAR_INDEXES stored procedure must run in the namespace of the independent ASP. Use the Set Auxiliary Storage Pool Group (SETASPGRP) command.

To remove an orphaned index from an independent ASP iaspXXX, you can use the following commands:

```
CL:
SETASPGRP(isapXXX)
SQL:
CALL SYSPROC.SYSTS_CLEAR_INDEXES( )
```

Note: If you use System i® Navigator, right-click the database name for the independent ASP, and run your SQL scripts.

Related concepts

[Server alias name](#)

You can use a server alias name to assign a meaningful name to a server.

Related reference

[QSYS2.SYSTEXTINDEXES administration table](#)

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

[Find orphaned and missing indexes](#)

You can find orphaned and missing indexes using an SQL User Defined Table Function (UDTF) named QDBTS_LISTINXSTS.

SYSPROC.SYSTS_VALIDITYCHECK

You can check for valid index items using the SYSPROC.SYSTS_VALIDITYCHECK SQL stored procedure.

Syntax

This stored procedure can fix some items that are not valid if the *autoFix* parameter is specified.

```
>>-SYSPROC.SYSTS_VALIDITYCHECK (indexSchema, indexName, autoFix) -><
```

The schema qualifier is SYSPROC.

Parameters

indexSchema

Identifies the schema of the text search index. If this parameter is null, the value of the CURRENT SCHEMA special register for the invoker is used.

The data type of this parameter is VARCHAR(128).

indexName

Identifies the name of the text search index. The name of the text search index with the index schema uniquely identifies the full-text index in the DB2 subsystem. You must specify a value that is not null for this parameter.

The data type for this parameter is VARCHAR(128).

autoFix

Identifies whether automatic fix is required. The value for this parameter can only be 0 or 1. The meanings of these values are described as follows:

0

Only the index validity is checked.

1

Index validity is checked and items that are not valid are fixed.

Note:

If values other than 0 or 1 are specified, they are considered as 0.

The data type for this parameter is INTEGER.

Restrictions: If `indexSchema` and `indexName` are both specified as `*NONE`, then the stored procedure checks only the validity for common parts of the product.

SQL for SYSTS_VALIDITYCHECK

```
CREATE PROCEDURE SYSPROC.SYSTS_VALIDITYCHECK
  (IN INDEXSCHEMA VARCHAR(128) CCSID 1208,
   IN INDEXNAME VARCHAR(128) CCSID 1208,
   IN AUTOFIX INTEGER)
  EXTERNAL NAME QDBTSLIB.DSN5VALCHK
  DYNAMIC RESULT SETS 0
  LANGUAGE C
  PARAMETER STYLE SQL
  MODIFIES SQL DATA
  PROGRAM TYPE MAIN
  COMMIT ON RETURN NO
  INHERIT SPECIAL REGISTERS
```

Examples

- To check the validity for an index, type the following command from any SQL interface:

```
CALL SYSPROC.SYSTS_VALIDITYCHECK('indexSchema1', 'indexName1', 0)
```

- To check and fix an index automatically:

```
CALL SYSPROC.SYSTS_VALIDITYCHECK('indexSchema1', 'indexName1', 1)
```

QDBTSLIB.QDBTS_ROWS_STATUS

Users can check which documents were not indexed successfully using the `QDBTSLIB.QDBTS_ROWS_STATUS` SQL stored procedure after calling `SYSTS_UPDATE`.

With this stored procedure, users can get a result set which presents those documents which were not indexed successful. Or users can choose generating a new table to store related info of those documents which were not indexed successful.

Syntax

```
>>-QDBTSLIB.QDBTS_ROWS_STATUS (IndexSchema, IndexName, ResultSetIndicator,
TableSchema, TableName) -><
```

The schema qualifier is `QDBTSLIB`.

Parameter

Required Parameter Group

IndexSchema

Identifies the schema of the text search index. You must specify a value that is not null for this parameter.

The data type of this parameter is `VARCHAR(128)`.

IndexName

Identifies the name of the text search index. The name of the text search index with the index schema uniquely identifies the full-text index in the DB2® subsystem. You must specify a value that is not null for this parameter.

The data type for this parameter is `VARCHAR(128)`.

Optional Parameter Group

ResultSetIndicator

Identifies whether return the result set to user directly or not.

If the ResultSetIndicator is not passed or it's set to zero, the result set is returned to the caller.

If the ResultSetIndicator is specified and is NOT set to zero, no result set is returned and the caller can query the table specified by TableSchema and TableName.

The data type for this parameter is INTEGER.

TableSchema

Identifies the table schema of the table. If ResultSetIndicator is specified and is NOT zero, then this schema must exist.

TableName

Identifies the table name generated by this stored procedure. If ResultSetIndicator is specified and is NOT zero, this stored procedure will create a new table with this table name.

Result Set or Table Structure

TIME TIMESTAMP
STATUS INTEGER
TEXT_STA VARCHAR(50)
TUS
MESSAGE VARCHAR(1024)
KEYCOLU Depends on the key columns defined in base table
MNNAME
S

TIME

This is the time when error/warning was thrown out. ...

STATUS, TEXT_STATUS

30 WARNIN this record was indexed but there is warning about it
 G
40 ERROR this record was not indexed successful for some errors
50 FATAL this record returns a fatal error and breaks up the indexing
 ERROR

MESSAGE

This shows the error code and error message. According to this column, users can know why the record was not indexed successful.

KEYCOLUMNAMES

There might be more than one key columns specified. If so, all the key columns will be returned. Each key column name will have a prefix 'PK_'.

Examples

- To check if the index has document not indexed successful:

```
CALL QDBTSLIB.QDBTS_ROWS_STATUS('indexSchema1','indexName1')
```

The result set will be returned to caller directly.

- To generate a new table to stored the result:

```
CALL QDBTSLIB.QDBTS_ROWS_STATUS('indexSchema1','indexName1',1,'TableSchema','ResultTable')
```

Then users can query from the result table to get more info.

```
SELECT * FROM TABLESCHEMA.RESULTTABLE;
```

Assume there are 2 columns (K1, K2) of the base table to be indexed.

To get the rows not indexed of base table, users can use following SQL statement.

```
SELECT b.*,r.MESSAGE FROM TABLESCHEMA.RESULTTABLE r LFET JOIN BASETABLESCHEMA.BASETABLE b on  
r.PK_K1=b.K1 and r.PK_K2=b.K2;
```

Then users can update that row based on the message returned, then invoke SYSTS_UPDATE again to index the new changed row.

Synonym dictionaries

A synonym dictionary can improve the quality of search results.

You can add a synonym dictionary to a collection at any time.

A synonym dictionary consists of synonym groups that you define in an XML file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>  
<synonymgroups version="1.0">  
<synonymgroup>  
  <synonym>Paixão</synonym>  
  <synonym>amor</synonym>  
  <synonym>flor</synonym>  
  <synonym>linda</synonym>  
</synonymgroup>  
<synonymgroup>  
  <synonym>worldwide patent tracking system</synonym>  
  <synonym>wpts</synonym>  
</synonymgroup>  
</synonymgroups>
```

Add a synonym dictionary to a collection

Specifying the synonym groups in a synonym dictionary improves the quality of text search results. The OMNIFIND administrator has the correct authority and privileges to run the synonym tool and IBM Navigator for i.

- To add a synonym dictionary to a collection with the synonym tool , follow these steps:
 - a) Create a synonym XML file by specifying the synonym groups, as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>  
<synonymgroups version="1.0">  
<synonymgroup>  
  <synonym>Paixão</synonym>  
  <synonym>amor</synonym>  
  <synonym>flor</synonym>  
  <synonym>linda</synonym>  
</synonymgroup>  
<synonymgroup>  
  <synonym>worldwide patent tracking system</synonym>  
  <synonym>wpts</synonym>  
</synonymgroup>  
</synonymgroups>
```

- b) Copy the synonym XML file to any directory on the text search server.
- c) Use the synonym tool to add the synonym dictionary to a collection.

You can add a synonym dictionary in append mode or replace mode. If you add a synonym dictionary in append mode, the new synonyms are added to the existing synonym dictionary. If you add a synonym dictionary in replace mode, the existing synonyms are replaced by the new synonyms that you defined for the text search index.

Option	Description
On IBM i, enter the following command (within the QSH interface):	<pre>synonymTool.sh importSynonym -synonymFile <absolute path to synonym XML file> -collectionName <collection name> -replace <[true false]> -configPath <absolute path to configuration folder></pre>
On a Linux server, enter the following command:	<pre>synonymTool.sh importSynonym -synonymFile <absolute path to synonym XML file> -collectionName <collection name> -replace <[true false]> -configPath <absolute path to configuration folder></pre>
On a Windows server, enter the following command:	<pre>synonymTool.bat importSynonym -synonymFile <absolute path to synonym XML file> -collectionName <collection name> -replace <[true false]> -configPath <absolute path to configuration folder></pre>

If the format of the XML file is not valid, or if the XML file is empty, an error code is returned.

- To import synonym dictionary to a collection from IBM Navigator for i, follow these steps.
 - a) From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
 - b) On the right panel, select **System > OmniFind > Collection List**.
 - c) Right click the collection and select **Import Synonym Dictionary**.

Remove a synonym dictionary from a collection

Use the script that is provided to remove a synonym dictionary from a collection.

The OMNIFIND administrator needs to retrieve the name of the collection from which you want the synonym dictionary to be removed.

Run the script to remove the synonym dictionary from a collection.

Option	Description
On IBM i, enter the following command (within the QSH interface):	<pre>removeSynonym.sh -collectionName <collection name> -configPath <absolute path to configuration folder></pre>
On a Linux server, enter the following command:	<pre>removeSynonym.sh -collectionName <collection name> -configPath <absolute path to configuration folder></pre>
On a Windows server, enter the following command:	<pre>removeSynonym.bat -collectionName <collection name> -configPath <absolute path to configuration folder></pre>

If a database has several text search indexes, you must complete this task for each of the corresponding collections.

Find orphaned and missing indexes

You can find orphaned and missing indexes using an SQL User Defined Table Function (UDTF) named QDBTS_LISTINXSTS.

An index can be orphaned if a SYSTS_DROP stored procedure is called and the server is stopped at the time the procedure is running.

The QDBTS_LISTINXSTS function combines all the integrated-file-system collections and catalog indexes in the current namespace into one table. The function decides which independent auxiliary storage pool (ASP) or *SYSBASE is set. It then scans the collection directory of each server in the independent ASP or *SYSBASE.

For *SYSBASE, each server directory under /QOpenSys/QIBM/ProdData/TextSearch is checked. For independent ASPs, each server directory under /the ASP number/QOpenSys/QIBM/ProdData/TextSearch is checked. For example, if the independent ASP number is 67, each server directory under /67/QOpenSys/QIBM/ProdData/TextSearch is checked.

For catalog index information, data is obtained from catalog table QSYS2.SYSTEXTINDEXES. If you want to check servers on an independent ASP, issue the Set Auxiliary Storage Pool Group (SETASPGRP) command before this function is called.

If you want to remove possible orphaned indexes from the integrated file system after they are identified, use the SYSPROC.SYSTS_REMOVE or SYSPROC.SYSTS_CLEAR_INDEXES stored procedure or the [“Advanced administration”](#) on page 84 (adminTool.sh).

Terms

Orphaned index

A collection (an index) exists in the integrated file system directory of the server, but no corresponding index is recorded in catalog QSYS2.SYSTEXTINDEXES.

Missing index

Index records exist in catalog QSYS2.SYSTEXTINDEXES, but the corresponding collection directory does not exist.

Syntax

```
>>-QDBTS_LISTINXSTS(--null--)->>
```

Return format

The QDBTS_LISTINXSTS function returns information of detected indexes in a table. See the following SQL command that is used to create the UDTF.

SQL for LISTINXSTS UDTF

```
CREATE FUNCTION QDBTSLIB.QDBTS_LISTINXSTS()  
  RETURNS TABLE(COLLECTIONNAME VARCHAR(255),  
                INDEXID INTEGER,  
                INDEXSCHEMA VARCHAR(128),  
                INDEXNAME VARCHAR(128),  
                SERVERID INTEGER)  
  SPECIFIC qdbts_listinxsts  
  SCRATCHPAD  
  NO FINAL CALL  
  LANGUAGE C++  
  PARAMETER STYLE DB2SQL  
  EXTERNAL NAME 'QDBTSLIB/QDBTSSP(checkIndex)';
```

Examples

- Detect all orphaned indexes:

```
SELECT COLLECTIONNAME, SERVERID
      FROM TABLE(QDBTSLIB.QDBTS_LISTINXSTS()) AS T
      WHERE T.INDEXSCHEMA IS NULL AND T.INDEXNAME IS NULL
```

- Detect all missing indexes:

```
SELECT INDEXSCHEMA, INDEXNAME
      FROM TABLE(QDBTSLIB.QDBTS_LISTINXSTS()) AS T
      WHERE T.COLLECTIONNAME IS NULL
```

- Detect orphaned indexes in serverid = 2 on the independent ASP iaspXXX:

CONNECT TO iaspXXX

SQL:

```
SELECT T.COLLECTIONNAME, S.SERVERPATH
      FROM TABLE(QDBTSLIB.QDBTS_LISTINXSTS())
           AS T LEFT OUTER JOIN QSYS2.SYSTEXTSERVERS S ON (T.SERVERID = S.SERVERID)
      WHERE T.INDEXSCHEMA IS NULL AND T.INDEXNAME IS NULL AND T.SERVERID = 2
```

Related reference

SYSPROC.SYSTS_DROP

You can call the SYSPROC.SYSTS_DROP stored procedure to drop a text search index that was defined by using the SYSPROC.SYSTS_CREATE stored procedure.

SYSPROC.SYSTS_REMOVE

You can remove orphaned indexes with the SYSPROC.SYSTS_REMOVE SQL stored procedure.

SYSTS_CLEAR_INDEXES

You can remove orphaned indexes with the SYSPROC.SYSTS_CLEAR_INDEXES SQL stored procedure. Another implicit way is invoking SYSTS_START directly, which tries to clear orphaned indexes automatically.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

Advanced administration

You can use the administration tool for advanced administration.

The OmniFind Text Search Server for DB2 for i can be running when you use the administration tool.

You can use the administration tool to do the following tasks:

- Check the status of collections, such as finding out how many documents are present
- Delete orphan collections
- Report the version of the server
- Report all the collections that are on the text search server

Commands

The command that you issue to run the administration tool depends on what operating system the text search server is installed on. The command also depends on the task that you want to do.

Table 31. Commands to check the status of collections and to delete orphaned collections

On IBM i (within the QSH interface)	On a Linux server	On a Windows server
<pre>adminTool.sh -[delete status] -collectionName <collection name> -configPath <absolute path to configuration folder></pre>	<pre>adminTool.sh -[delete status] -collectionName <collection name> -configPath <absolute path to configuration folder></pre>	<pre>adminTool.bat -[delete status] -collectionName <collection name> -configPath <absolute path to configuration folder></pre>

Table 32. Commands to display the version of the server and to report all the collections

On IBM i (within the QSH interface)	On a Linux server	On a Windows server
<pre>adminTool.sh -[version] -configPath <absolute path to configuration folder></pre>	<pre>adminTool.sh -[version reportAll] -configPath <absolute path to configuration folder></pre>	<pre>adminTool.bat -[version reportAll] -configPath <absolute path to configuration folder></pre>

Options

status

Checks the status of the collection.

delete

Specifies that you want to delete the orphaned collection.

version

Displays the version of the server.

reportAll

Reports all the collections that are on the text search server.

Example

To find out the version of the server, enter the following command on a Linux server:

```
adminTool.sh -version -s <absolute path to server config.xml>
```

When you use a Windows server, a corresponding .bat script is provided.

ServerInstance tool

You can use the ServerInstance tool to create or delete servers on *SYSBASE or an independent auxiliary storage pool (ASP). You can also use the ServerInstance tool to link files from a server to the server where OmniFind Text Search Server for DB2 for i is installed.

By default, OmniFind Text Search Server for DB2 for i is installed under directory /Q0penSys/QIBM/ProdData/TextSearch/server1.

You can use the ServerInstance tool to complete the following tasks before you use it to stop server1 on *SYSBASE:

- Create a server on *SYSBASE or independent ASPs
- Delete a server on *SYSBASE or independent ASPs
- Link files from a server to server1

Syntax

```
ServerInstance.sh -[create|delete|relink]  
-servernum <server number>  
(-port <port>)  
(-device <device name>)
```

Command options

create

Creates a server.

delete

Deletes a server.

relink

Links files from a server to server1.

Note: You do not need this option after you have program temporary fix (PTF) SI31548 installed on your system. The system automatically processes the linking operation if you have this PTF installed.

Parameters

servernum

Specifies the server number. For example, when a server with server number 3 is created, the directory of the server is /QOpenSys/QIBM/ProdData/TextSearch/server3.

port

Specifies the port of the server. This parameter is needed only when you create a server.

device

Specifies the name of the independent ASP. This parameter is needed only when the operation is completed on the independent ASP.

Examples

- To create a server with server number 2 and port number 50000 on *SYSBASE:

```
ServerInstance.sh -create -servernum 2 -port 50000
```

- To create a server with server number 3 and port number 50001 on independent ASP iasp1:

```
ServerInstance.sh -create -servernum 3 -port 50001 -device iasp1
```

- To delete a server with server number 2 on *SYSBASE:

```
ServerInstance.sh -delete -servernum 2
```

- To delete a server with server number 3 on independent ASP iasp1:

```
ServerInstance.sh -delete -servernum 3 -device iasp1
```

- To link files from a server to server number 2 on *SYSBASE:

```
ServerInstance.sh -relink -servernum 2
```

- To link files from a server to server number 3 on independent ASP iasp1:

```
ServerInstance.sh -relink -servernum 3 -device iasp1
```

Health Checker

Health checker is an environment verification tool that can be used to diagnose any OmniFind Text Search Server for DB2 for i configuration problems. It can be used to verify that the OmniFind Text Search Server

for DB2 for i text servers and indexes are correctly functioning, and will generate a report with warnings or errors for any potential issues found.

Prerequisites

The OmniFind Text Search Server for DB2 for i health checker is based on ARE (Application Runtime Expert for i, product 5770-ARE). Therefore, ARE must be installed for this tool to run.

Health Checker Procedures

Health checker is invoked through a series of DB2 SQL stored procedures. The procedures can be invoked through any SQL interface, including from a high level language such as RPG and COBOL. All information, including warnings or errors, is returned through a corresponding result set.

SYSPROC.SYSTS_HC_GENERAL()

This stored procedure is used to check the general health of the OmniFind Text Search Server for DB2 for i product. The procedure checks the configuration of the product. This is an example to return all warning and error messages related to general OmniFind Text Search Server for DB2 for i configuration information:

```
> CALL SYSPROC.SYSTS_HC_GENERAL();
```

SYSPROC.SYSTS_HC_USR_AUTH()

This stored procedure is used to identify any authority issues for the invoking user that would prevent them from using the OmniFind Text Search Server for DB2 for i for searches. This is an example to return any warnings or errors regarding the invoking user's authorities:

```
> CALL SYSPROC.SYSTS_HC_USR_AUTH();
```

SYSPROC.SYSTS_HC_IDX()

This stored procedure is used to check all OmniFind Text Search Server for DB2 for i text indexes to ensure they are in a valid state. Note: If using an IASP (Independent Auxiliary Storage Pool) group, this only applies to the IASP group currently active in the job.

SYSSTS.HC_SVR()

This stored procedure is used to check the health of all local OmniFind Text Search Server for DB2 for i text servers that have been defined.

Additional Information

More information about the OmniFind Text Search Server for DB2 for i health checker, including numerous additional stored procedures options as well as a QShell interface, is available on developerWorks [OmniFind Text Search Server for DB2 for i under the article 'Health Checker'](#).

Independent ASP considerations for OmniFind Text Search Server for DB2 for i

You can administer a text search index on an independent auxiliary storage pool (ASP). The ASP can be switched between multiple systems, so there are additional considerations.

A local text search server is created during the installation of OMNIFIND. For independent ASPs, a local text server is created by an administrator using the ServerInstance tool (`ServerInstance.sh`) after the independent ASP group is created.

After you create a local text search server on the independent ASP, the index data exists on the independent ASP file system. The data is available if the independent ASP is switched to a different system. The administrator needs to create a local text search server only once for each independent ASP group.

Text search indexes that are on the independent ASP must be contained in text search servers that have been defined in the independent ASP. You cannot view a text search server defined in a different independent ASP group or in the system ASP when the job is connected to the independent ASP.

To create a text search server on an independent ASP named *myiasp*, follow these steps:

1. Vary on the independent ASP with the [Work with Configuration Status \(WRKCFGSTS\)](#) CL command or by using System i Navigator.
2. Connect to the namespace of the independent ASP group by using the [Set Auxiliary Pool Group \(SETASPGRP\)](#) CL command.
3. Use the `ServerInstance.sh` script to create a text search server.

Here is an example QSH command to use:

```
/QOpenSys/QIBM/ProdData/TextSearch/ServerInstance.sh -create  
-servernum 2 -port nnnnn -device myiasp
```

In the command, *nnnnn* is an available port number for the server to use. This port number must be available for use on all systems that the independent ASP group can be switched to.

After a text search server is defined for the independent ASP group, the administrative stored procedures can be used to start and stop the text search server. The stored procedures can also be used to create, drop, and update text search indexes.

Note: Job scheduler entries are added when the independent ASP is varied on for any indexes with scheduled updates that exist in the independent ASP. The job scheduler entries allow scheduled updates to continue, even when the independent ASP is switched between systems.

Restrictions of using text search indexes and independent ASPs

- All systems that the independent ASP can be switched between must have OmniFind Text Search Server for DB2 for i installed, and must be at the same program temporary fix (PTF) levels.
- Do not create text search indexes on an ASP other than the one that the table index is built over.
- The system catalogs SYSTEXTSERVERS, SYSTEXTINDEXES, SYSTEXTDEFAULTS, SYSTEXTCOLUMNS, and SYSTEXTCONFIGURATION do not contain records for indexes and servers that are defined in a different ASP group, including the system ASP. The catalogs contain rows only for indexes and servers that are defined for the independent ASP group that the job is connected to.
- The administrative stored procedures can be used to perform functions only on text search servers and indexes that are defined in the independent ASP group that the job is connected to.

Note: You can use the CONTAINS and SCORE SQL statements when a job is connected to an independent ASP group, even if the column is based on a table that exists on the system ASP.

High Availability

You can implement a high availability solution that includes text search indexes using existing APIs and commands. DB2 for IBM i now recognizes text search indexes and takes special actions during DB2 operations that affect these indexes.

Special considerations for text search indexes, high availability, and database administration

- A delete file (DLTF CL command) or DROP VIEW SQL statement against the view representing the text search index results in a drop of the text search index. The drop fails if the drop is executed under commitment control.
- Restoring a table or physical file that was saved with a text search index over a column does not enable the triggers that were created for the text search index before the save. If the text search index is later

restored or created, triggers are added as part of the index creation. This method enables applications to work, even if the text search index originally built over the table is not restored or created.

Replaying journal entries for OmniFind Administrative stored procedures

- A replay of the journal entry (QDBRPLAY API) for the creation of the view that represents the text search index results in creating the text search index.
- The addition and removal of the triggers on the based-on table during SYSTS_CREATE or SYSTS_DROP procedure calls does not cause journal entries to be logged.
- A call to the SYSTS_UPDATE or SYSTS_REPRIMEINDEX stored procedure results in a journal entry being logged against the view of the text search index. A replay of this journal entry using the QDBRPLAY API results in the correct procedure call being replayed.
- Automatically scheduled updates do NOT result in journal entries for the index and cannot be replayed.
- A call to SYSTS_ALTER results in a change journal entry against the view. A replay of the journal entry using the QDBRPLAY API causes the SYSTS_ALTER call to be replayed.

High Availability recommendations

If you want to implement a high availability solution, consider the following recommendations:

- After Creating the text search index on the primary system, the QDBRPLAY API can be used to replay the create on the backup system.
- The backup system now contains a duplicate index, with scheduled updates that occur at the same frequency as the primary system.
- As record changes are replayed into the backup systems based on table, they are logged in the staging table of the backup system.
- Calls to SYSTS_UPDATE, SYSTS_REPRIMEINDEX, and SYSTS_ALTER on the primary system results in journal entries that can be replayed on the backup system.
- It is NOT necessary for customers to journal and replicate the IFS files for the Text Search indexes.
- It *is* necessary to ensure that the triggers that have been added as part of SYSTS_CREATE are enabled to record changes to the based on table. These triggers can be identified by using the special "QDBTS" prefix on the name of the trigger.

Apply Journal Changes CL command (APYJRNCHG)

The Apply Journal Changes (APYJRNCHG) CL command can be used to replay OmniFind events. Users wanting to use this command must carefully consider the order in which journal entries are replayed. The staging table in QSYS2 must be exactly synchronized with the based-on table at the time a SYSTS_UPDATE is replayed or invoked.

Record changes that are applied to the based-on table are NOT logged in the staging table during the APYJRNCHG process. In some cases, it might be necessary to call the SYSTS_REPRIMEINDEX stored procedure to rebuild the index after using this command.

Related reference

SYSPROC.SYSTS_REPRIMEINDEX

You can reprime the index and start an initial update using the SYSPROC.SYSTS_REPRIMEINDEX stored procedure. Use this stored procedure when you want to restore data from the base table.

Performance analysis

OMNIFIND performance analysis includes choosing the correct index definition, handling documents efficiently, and specifying a selective search.

Background

OmniFind processing is a combination of work on a front end 'client' job and work on a backend 'server' job, with communication occurring between these jobs. Communication is performed using standard socket connections.

The client job reads records from the appropriate database table during index builds and maintenance. It processes the log of table changes, sends documents for ingestion, and handles any text search requests using the CONTAINS or SCORE SQL functions. The client job is the one in which the procedure, for example, SYSTS_CREATE or SYSTS_UPDATE, or the query with CONTAINS or SCORE, is run. Client job performance is dependent on processing database actions quickly, and retrieving and transporting text documents efficiently to the server job.

The server job parses documents sent from the client job, maintains the associated text index with inserts and deletions, and handles search requests within the index. The performance of the server job is dependent on its ability to communicate efficiently with the client jobs and to handle documents efficiently. There is normally a single server job serving multiple client jobs. Consequently, the server job is a multi-threaded job so that it can handle multiple clients.

A text index is not updated immediately with any changes to the underlying database table. Instead, any record changes to the table are logged using a combination of a database trigger and a staging table. The staging table records the update type (insert, update, or delete) along with an indication of which record in the base table was changed. The changed text is not captured in the log; only information to identify the record that was changed.

Table record changes are not reflected in the text index, and consequently not in searches with CONTAINS or SCORE, until the next successful SYSTS_UPDATE.

Subsequent updates to a text index after the initial update are called incremental updates. These updates add or delete documents to the text index based on any record changes made to the underlying table since the last update.

Any changes made to the base table are registered in a staging table. On an incremental update, the staging table is used to determine which records in the base table have changed. Those records are later read from the base table and their updated contents are reflected in the index.

The initial update is a more efficient process, per row, than an incremental update. The base table is processed without needing to also process the staging table. Therefore, a good performance technique is to do the initial update on the text index after the underlying table has been initially populated. This technique minimizes the time to populate the documents into the index.

An important performance-related configuration option is UPDATEAUTOCOMMIT. This value defines how frequently the database client job interrupts document processing. The client waits for the server job to confirm that it has processed all documents currently sent to it. UPDATEAUTOCOMMIT is used as a checkpoint method to allow the database to set boundaries of completed work.

If the index update is interrupted and continued later, the process restarts at the checkpoint boundary. As with any interruption, a checkpoint boundary forces the flow of documents to be suspended and the pipeline between the client and server to be cleared of documents, or in other words, emptied of documents. This start and stop process can have considerable negative effect on performance.

The default value for UPDATEAUTOCOMMIT is 100, which provides frequent checkpoints. Setting the value higher usually results in better response time performance for SYSTS_UPDATE calls. However, the higher value does mean a longer recovery time if the update is canceled and restarted.

Setting UPDATEAUTOCOMMIT to a large number (or zero, which means no checkpoint occurs) provides the best response time performance. However, if the initial update is canceled, OmniFind must start over from the beginning of the index build because there was no checkpoint.

For some customers, setting the value 5000 - 20000 appears to provide a reasonable balance between performance and checkpoint recoverability.

Choose the correct index definition

A text index can be specified with one of four FORMAT configuration types: TEXT, HTML, XML, and INSO.

1. TEXT is usually the most efficient format. The text is read from the database record and sent to the text server and the server processes it directly.

2. HTML is used when the documents are known to be in the form of HyperText Markup Language. The text is processed with consideration of ignoring markup control values within the document.
3. XML is used when the documents are known to be in the form of eXtensible Markup Language. Special consideration of the structural nature of the document contents is done, with tracking of elements, attributes, and the hierarchy within the document. Marking an index as XML provides the ability to do XML searches using the xmlxp (xpath) search language on the CONTAINS and SCORE functions.
4. INSO is used when the document needs to be processed INSIDE Out. In this form, the contents of each document is assumed to be more than simple text. Each document is pre-analyzed to determine which type of document it is, then converted to plain text. INSO documents are usually rich text word-processing documents generated by word-processing programs.

From a performance perspective, the work to index a document increases as you move from format TEXT to format INSO. While format INSO handles simple text, it is more overhead than format TEXT, as interpretation takes place. Use TEXT when the document is simple text in the database column.

The LANGUAGE configuration option is not strictly required, as OmniFind determines the language of the document based on examination. However, if the language of the documents is known, specifying it on the LANGUAGE option can speed up performance. It limits the amount of interpretation that needs to be done.

Handle documents efficiently

The client and server jobs communicate the document information using UTF (Unicode Transformation Format). To get the document into this format, the client reads the document from the database, converting the resulting data into UTF-8, also known as CCSID 1208. All text is converted to UTF-8 before being sent to the server.

To improve performance, set the text column being indexed to CCSID 1208 to avoid this conversion, improving the efficiency of the document handling process.

Use SYSTS_REPRIMEINDEX instead of SYSTS_UPDATE for a possible performance improvement

SYSTS_REPRIMEINDEX and SYSTS_UPDATE are used to update the text search index. SYSTS_REPRIMEINDEX is used to recreate the index. SYSTS_UPDATE is used to update the index with the incremental changes used after last successful update. In some instances, SYSTS_REPRIMEINDEX may perform better than the SYSTS_UPDATE.

To determine if the SYSTS_REPRIMEINDEX will perform better than the SYSTS_UPDATE, check the column, PENDINGCOUNT of Text Search Index view. The PENDINGCOUNT column will indicate how many rows will need to be changed for the next update process. If the user updated the same record in base table 10 times, the PENDINGCOUNT column will be increased by 10. Therefore, if the user is updating the base table frequently, the PENDINGCOUNT column will be increased. In most cases, if the PENDINGCOUNT column is larger than the total count of the base table, then the SYSTS_REPRIMEINDEX procedure will be the better choice to use rather than the SYSTS_UPDATE.

Specify a selective search

Text searches are done using the CONTAINS or SCORE functions within an SQL query statement. These searches compare the function search criteria to the documents associated with the column being searched. Matching records are identified and selected. The client job sends the search request to the server and receives the answer on whether a match is found.

As with all search criteria, the more selective the search, the more efficient the search is. Looking for common phrases, such as 'the' in the English language, results in numerous matches and can negatively affect performance. Practically speaking, such a search is unlikely to provide meaningful information. Specifying more selective search phrases results in fewer, more meaningful matches.

When using CONTAINS in the WHERE clause of an SQL statement, it usually performs best to have it ANDed to other criteria. For example:

```
SELECT bn, pubdate, description
FROM myBooks
WHERE CONTAINS(description,'Alladin') = 1
AND Pubdate > '2004-01-01'
```

Using CONTAINS and SCORE

The optimizer can improve the performance of the CONTAINS and SCORE functions by internally combining and replacing these built-in functions with a user-defined table function (UDTF). The UDTF returns a list of matching documents in one result. This UDTF processing in many cases performs better than the alternative process of invoking the built-in function for each record to determine a match.

To enable the optimizer to perform the UDTF rewrite, the CONTAINS function must:

- reside in the WHERE clause of the SQL statement
- be connected by "AND" to every additional predicate in the WHERE clause
- be a comparison with the value 1.

For example, the clause:

```
WHERE CONTAINS(MyDocuments, 'java performance') = 1
```

could be rewritten by the optimizer in the UDTF form. However, the clause:

```
WHERE CONTAINS(MyDocuments, 'java performance') = 1 OR price >100
```

could not be rewritten due to the CONTAINS being ORed to other predicates.

Note: when a UDTF rewrite is enabled, the optimizer still uses cost comparison to choose the optimal plan.

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

SYSPROC.SYSTS_UPDATE

You can call the SYSPROC.SYSTS_UPDATE stored procedure to update the text search index to reflect the current contents of the text column.

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

Transaction considerations

Consider your environment when deciding how often to update your text indexes from the underlying data in the database tables.

A large consideration for traditional database users is the concept of transaction boundaries and transaction processing. A classic example is a bank transaction where money is transferred from one account to another. The transfer is considered to be a single transaction; either the transfer occurs or it does not. The customer would not appreciate having the money removed from one account but not show up in the other account until some time later. Conversely, the bank would not want the money to show up in both accounts for some amount of time until the transaction was finally completed. The idea is that if a change is made, it is reflected immediately.

There are many examples in information management where delays are more acceptable and even expected. In a traditional data warehouse design, the contents of the data warehouse often lag the operational data by hours or more. This acceptance of delay is based on a combination of realistic expectations of a data warehouse and a desire for predictable, coherent data.

Unstructured text searches lie somewhere between these two boundaries. Practically speaking, indexing text documents is an intensive process to analyze and break down the underlying meaning of the words in the document. Text searches allow a search for 'mice' to find documents with 'mouse'. This search result happens due to the ability of the indexing technique to break down words into their underlying meanings. This analysis is done at the time when a document is indexed in order to make subsequent searches as fast as possible.

Users of a traditional database index expect the index to reflect the state of the data in the database table. This same expectation does not hold for a text index. The contents of the text index reflect the state of the table based on the time when the last update (SYSTS_UPDATE) was performed.

In a highly changing environment, it is unlikely that the text index would reflect the current state of the table at any given time. However, in a more predictable environment where the database table is updated less frequently or in batch mode, the text index updates can be timed to perform after the table update, accurately reflecting the state of the table.

It is important to have the appropriate expectation for a text index. Use the UPDATE FREQUENCY option on the text index "[SYSPROC.SYSTS_CREATE](#)" on page 17 or "[SYSPROC.SYSTS_UPDATE](#)" on page 32, or explicitly call the SYSTS_UPDATE procedure to update the contents of the text index appropriately.

For more static environments such as bulk data loads, it makes sense to time the text index update to take place after the bulk load is completed. For more transaction-oriented environments, the UPDATE FREQUENCY value can be set to a short duration or the SYSTS_UPDATE procedure can be invoked frequently. It is normally true that the more frequent the update, the more workload is placed on the machine.

Related reference

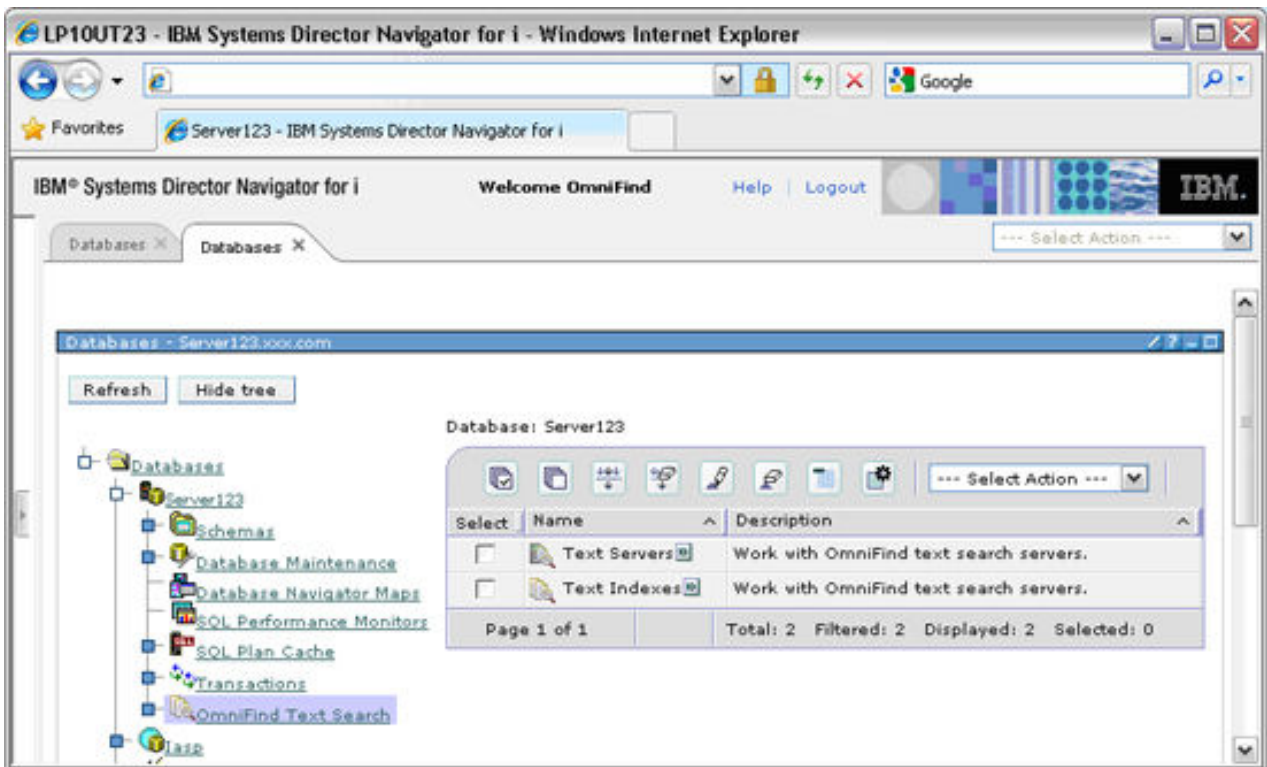
[QSYS2.SYSTEXTINDEXES administration table](#)

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

Using IBM Navigator for i

You can administer your OmniFind text search servers and text search indexes using IBM Navigator for i.

1. In the **IBM Navigator for i** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Select **OmniFind Text Search**.

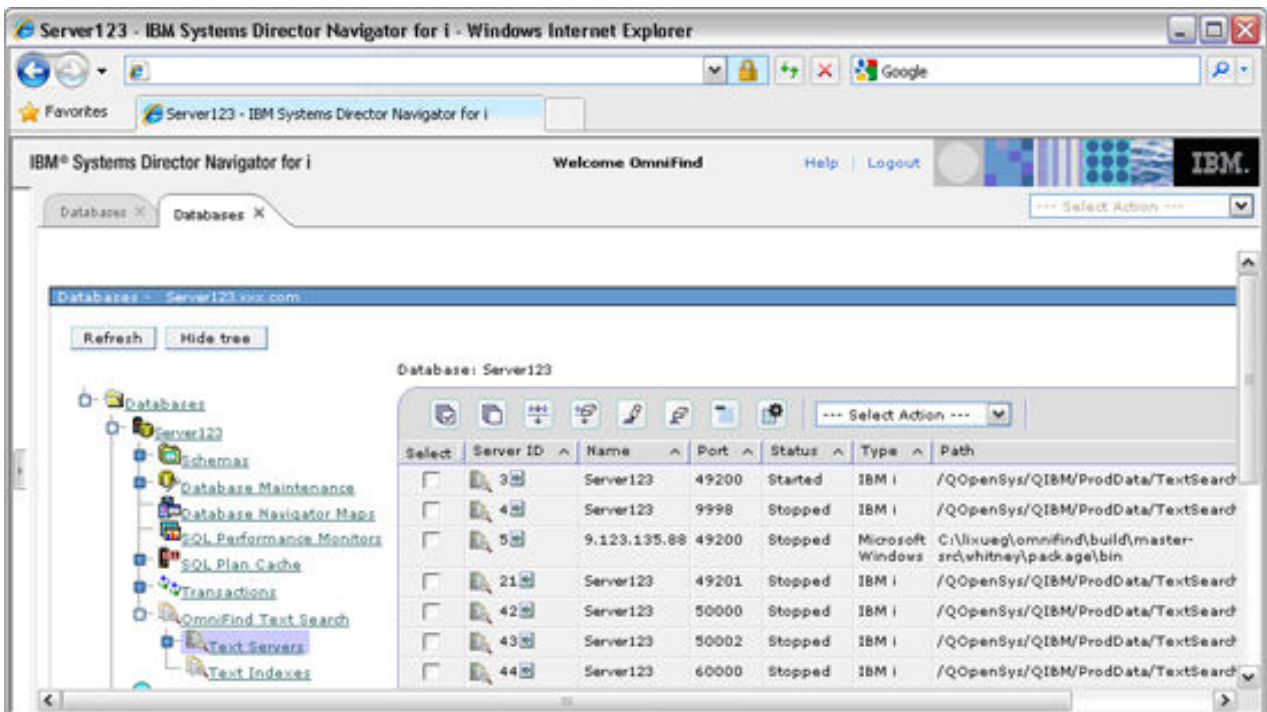


Work with text search servers

You can start and stop your OmniFind text search servers and create a text search index using IBM Systems Director Navigator for i.

Viewing the status of your OmniFind Text Servers:

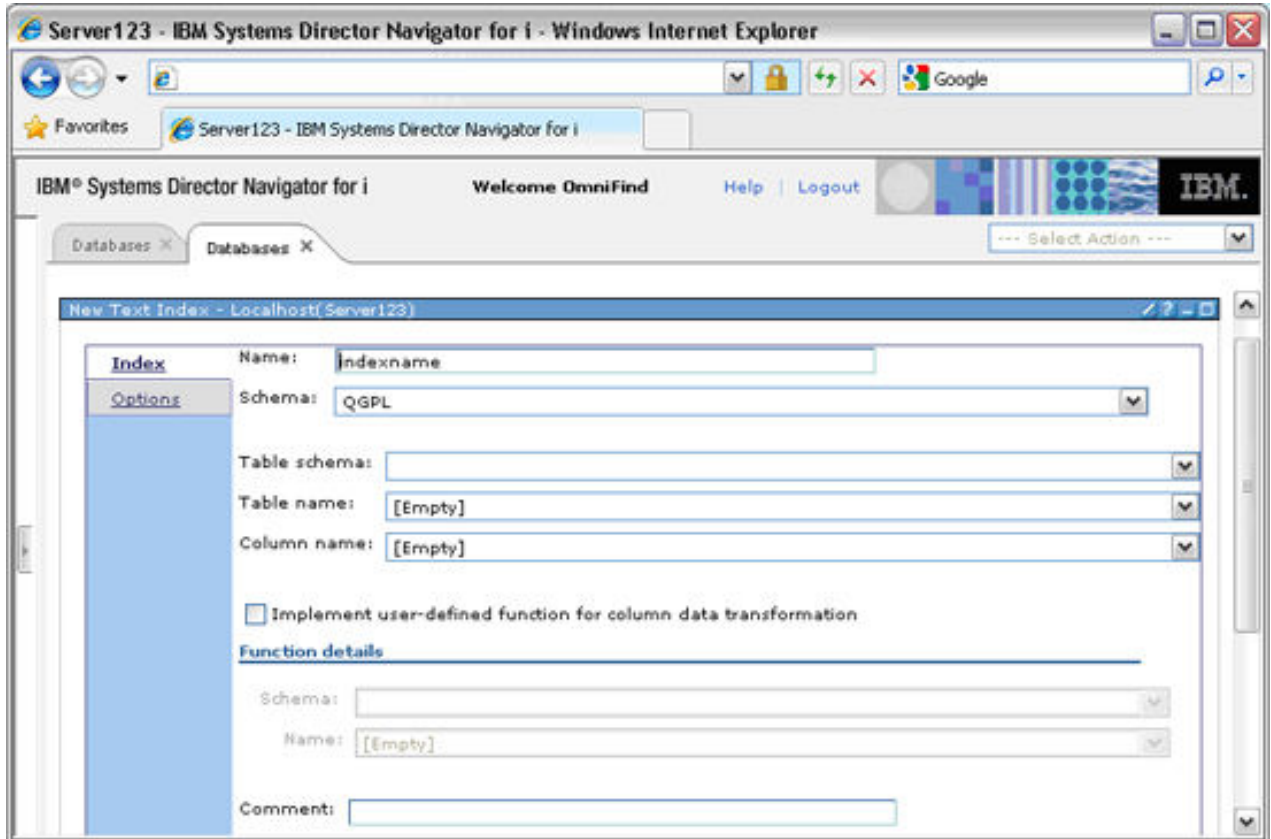
1. Select the Text Servers folder in the right pane.
2. View the status of the currently configured text search servers on the system in the right pane.
3. Select **Refresh** to refresh the list of servers.



Starting and stopping your OmniFind Text Servers:

1. Select the box in front of your selected Server ID in the right pane.
2. Select from the **Actions** menu:
 - Start server
 - Stop server
 - Create a text search index

Creating a text search index:

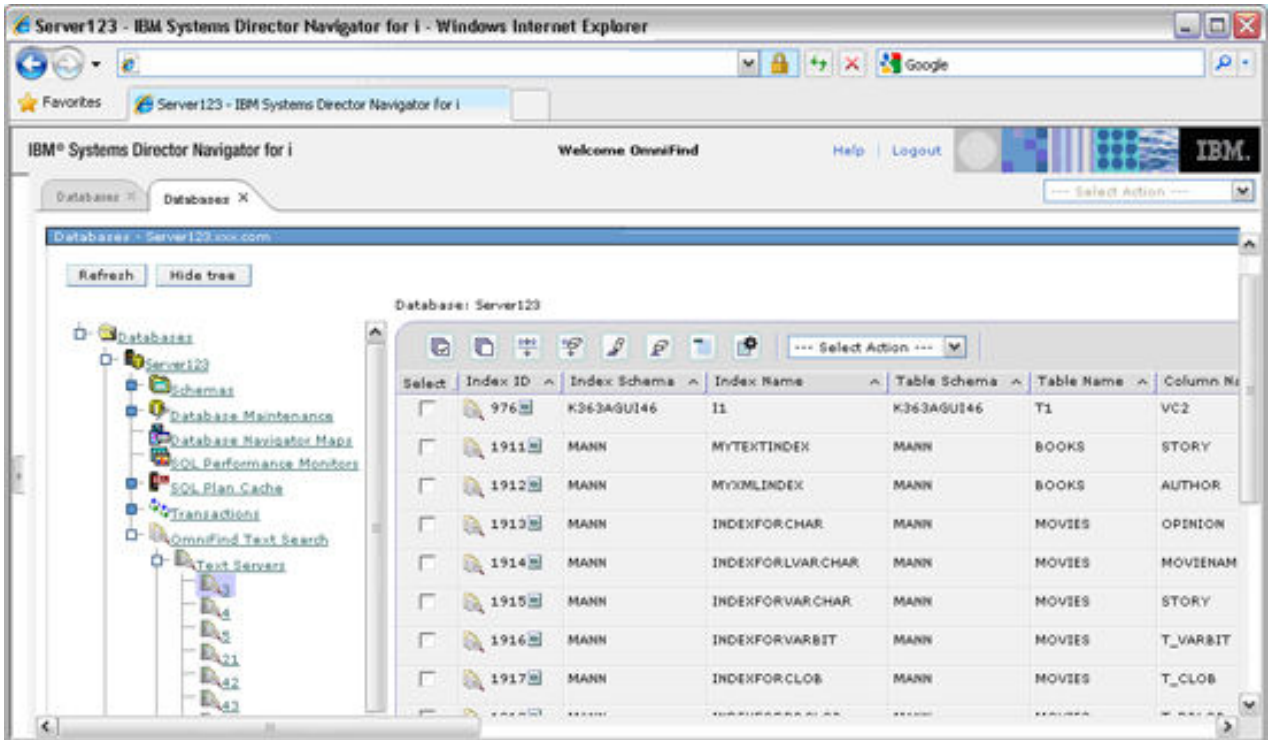


Work with text search indexes

You can perform operations on any text search index on a system using IBM Systems Director Navigator for i.

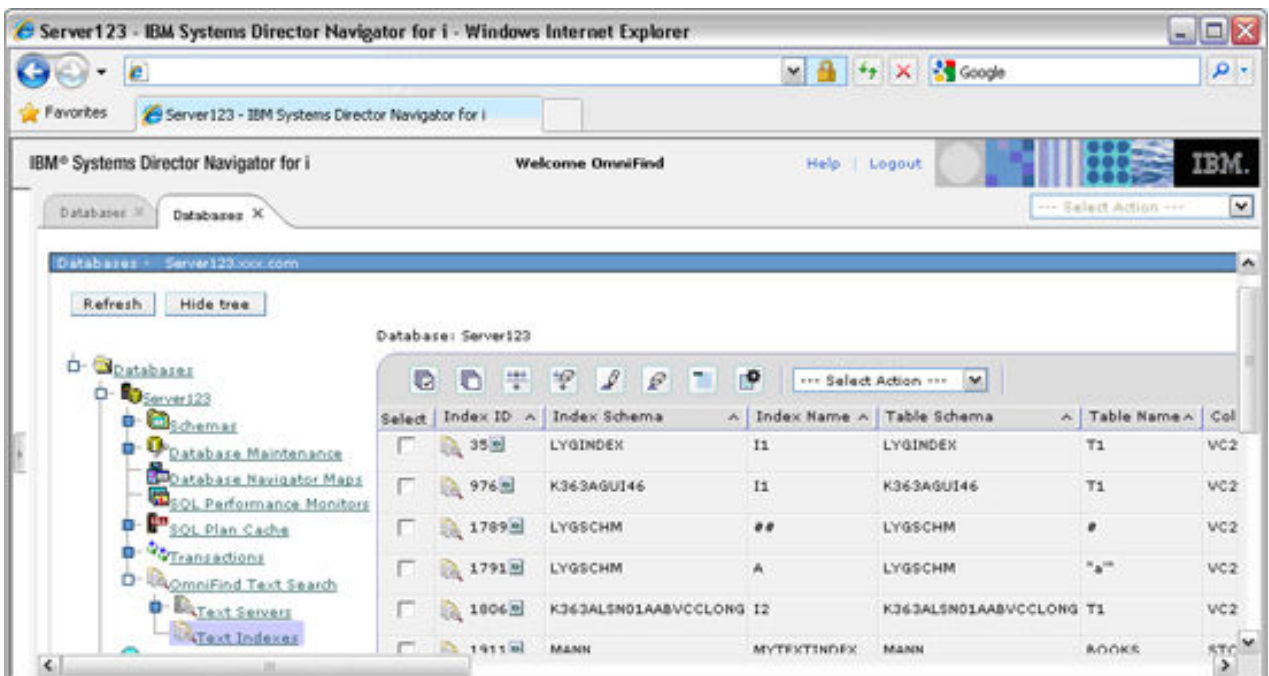
Viewing the OmniFind text indexes for a text server:

1. In the **IBM Systems Director Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Select **OmniFind Text Search**.
5. Select **Text Servers**.
6. Select the text server that you want to work with. The indexes for that server are displayed in the right pane.



Viewing the OmniFind text indexes for this system:

1. Select **OmniFind Text Search**.
2. Select **Text Indexes**. These indexes are all the text search indexes for this system in the current partition.



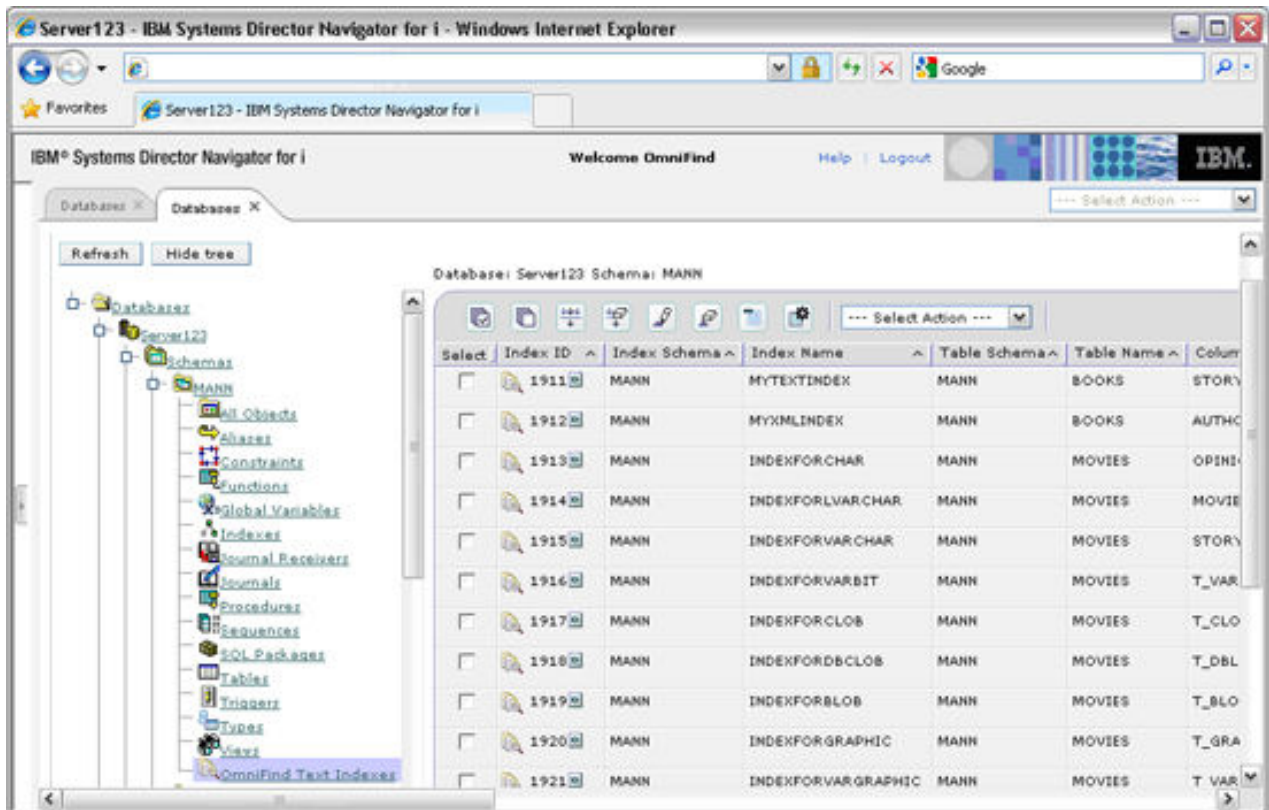
Perform operations on a text index:

1. Select the box for the index in the right pane that you want to work with.
2. **Select Action** from the box at the top of the panel.
 - Definition
 - Update

- Reprime
- Delete
- Description

Viewing the OmniFind text indexes for a schema:

1. In the **IBM Systems Director Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Expand **Schemas**.
5. Expand the schema that you want to work with.
6. Select **OmniFind Text Indexes**. These indexes are all the text search indexes for this schema.



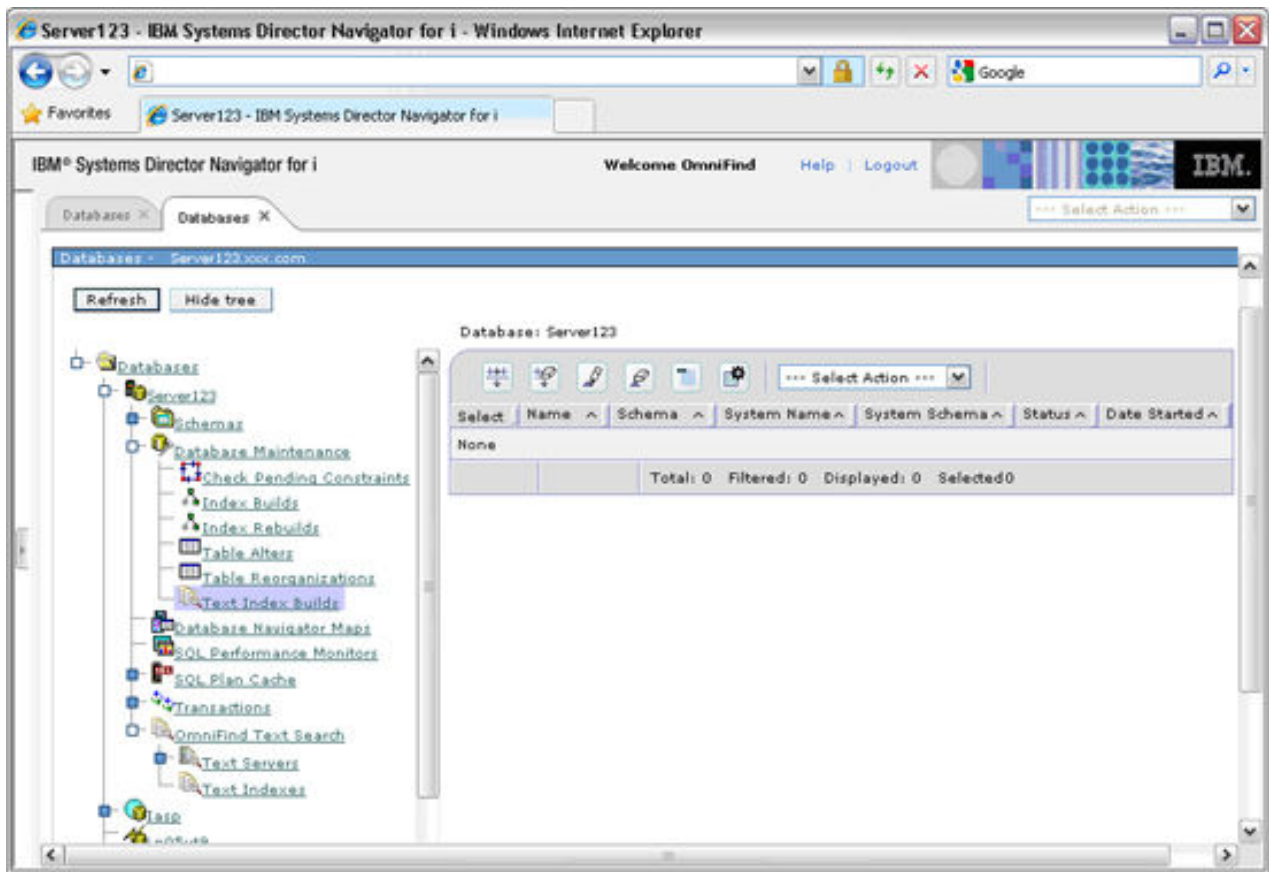
View text search index builds

You can view text indexes that are being built by the database using IBM Systems Director Navigator. This view is helpful in determining when text search indexes become available to your applications.

To display text search indexes that are being built, follow these steps:

1. In the **IBM Systems Director Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Select **Database Maintenance**.
5. Select **Text Index Builds**.

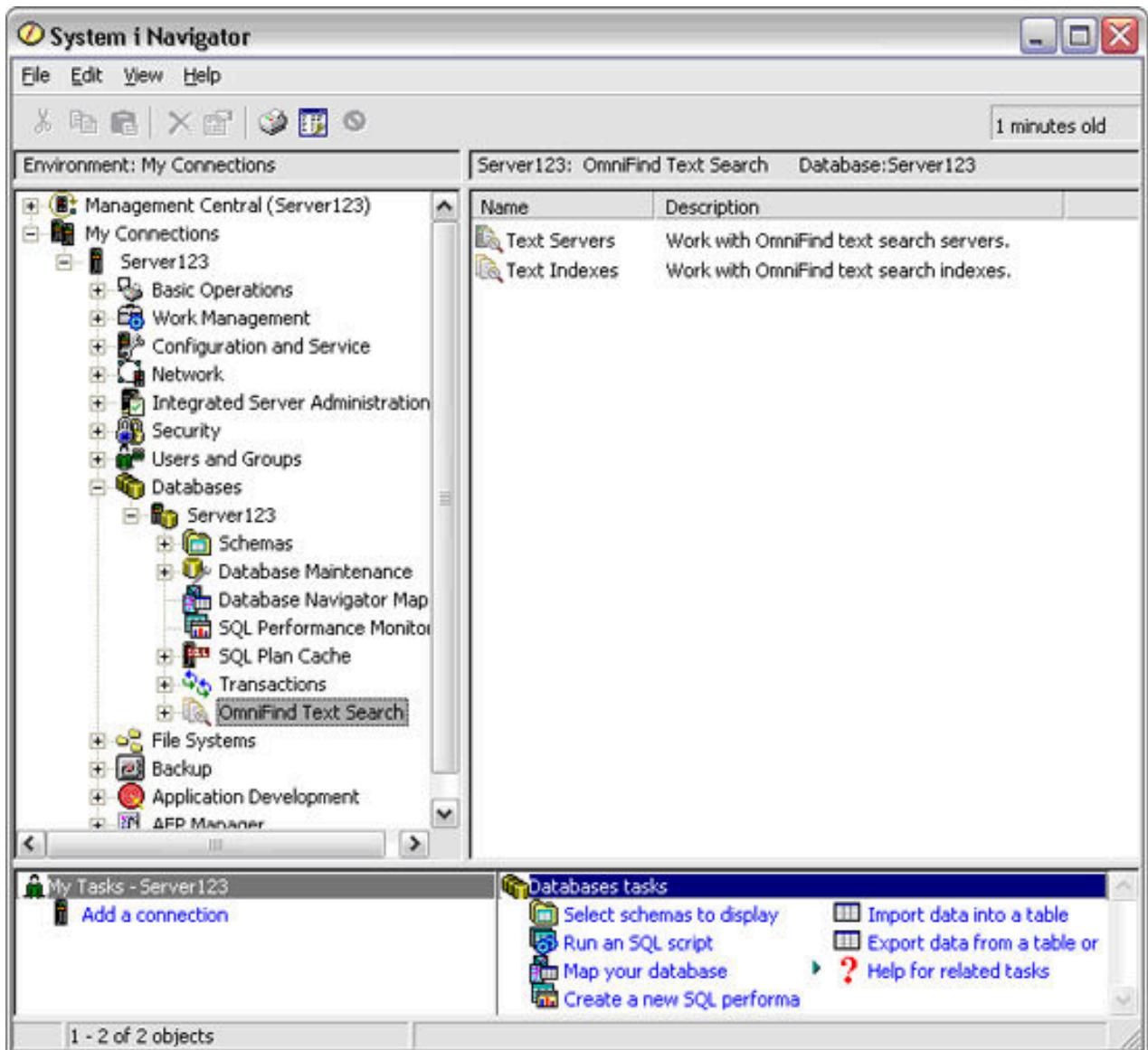
This panel shows only text search index updates in progress. It is empty if there are no updates or repriming currently running on the system.



Using System i Navigator

You can administer your OmniFind text search servers and text search indexes using System i Navigator.

1. In the **System i Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Select **OmniFind Text Search**.

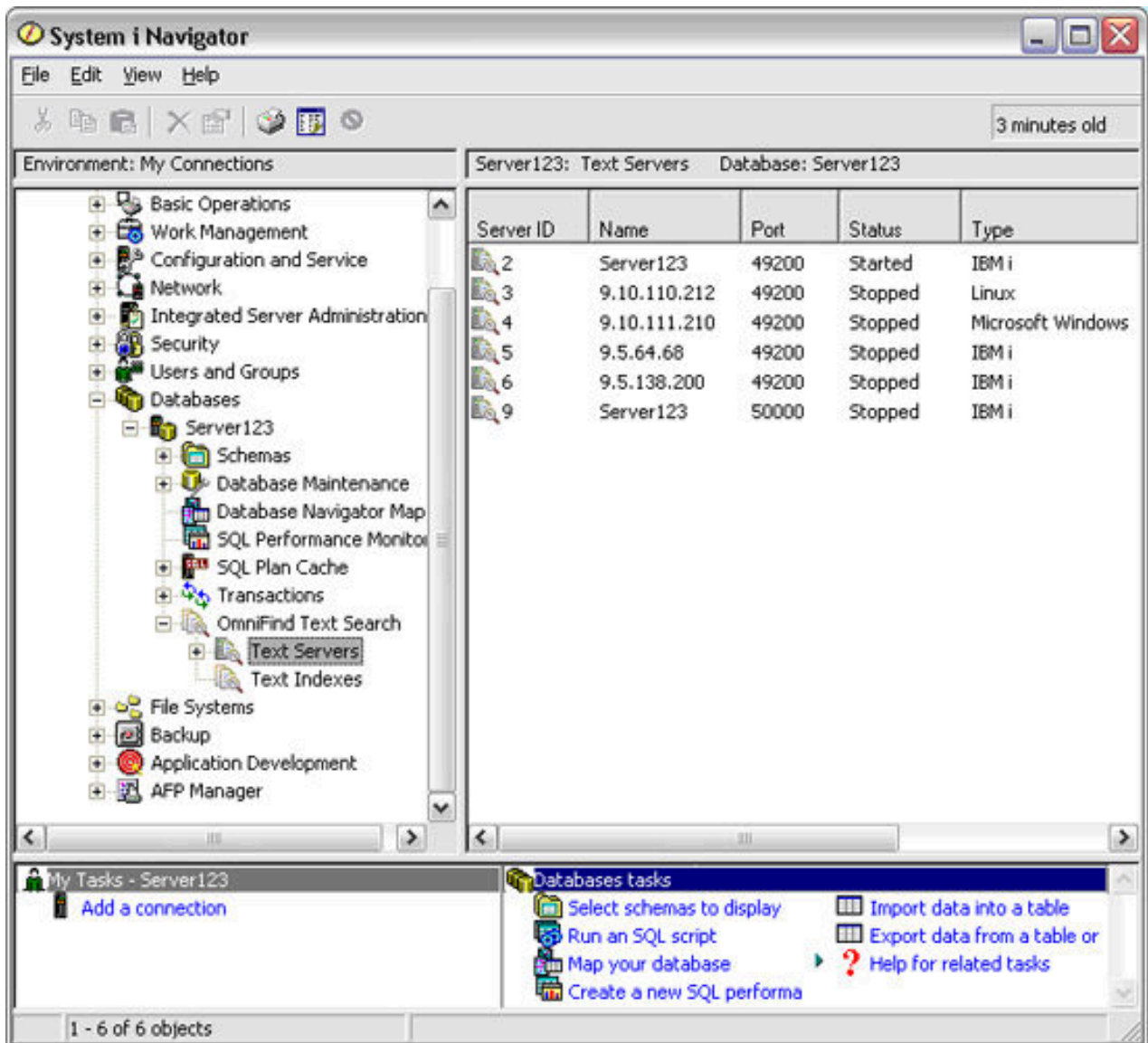


Work with text search servers

You can start and stop your OmniFind text search servers and create a text search index using System i Navigator.

Viewing the status of your OmniFind Text Servers:

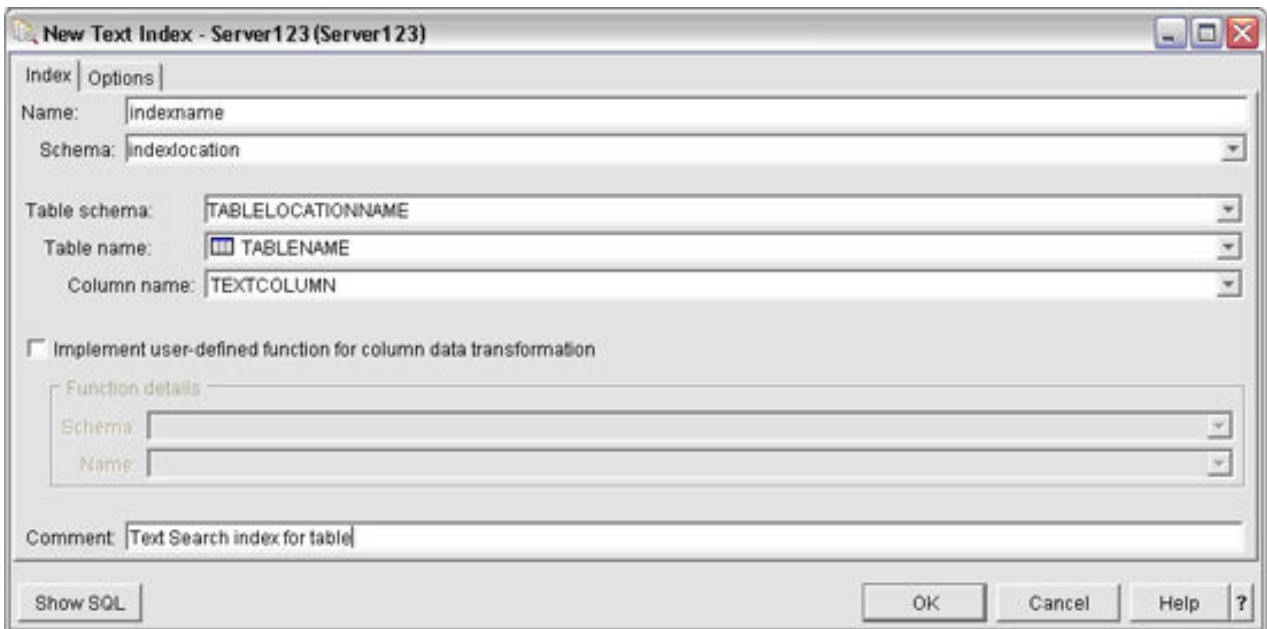
1. Select the Text Servers folder in the right pane.
2. View the status of the currently configured text search servers on the system in the right pane.
3. Select F5 to refresh the list of servers.



Starting and stopping your OmniFind Text Servers:

1. Select your selected Server ID in the right pane.
2. Right-click to view options:
 - Start server
 - Stop server
 - Create a text search index

Creating a text search index:

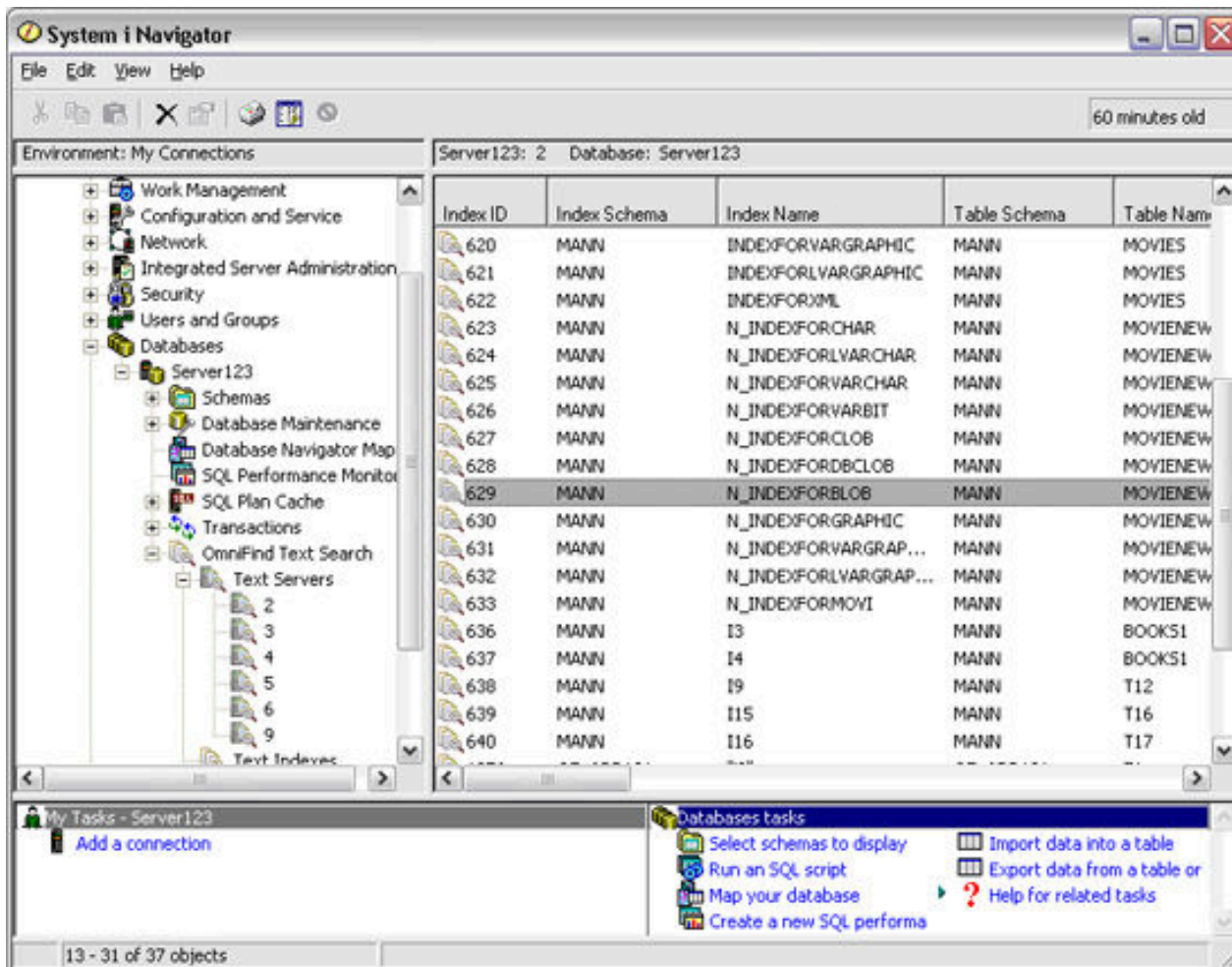


Work with text search indexes

You can perform operations on any text search index on a system using System i Navigator.

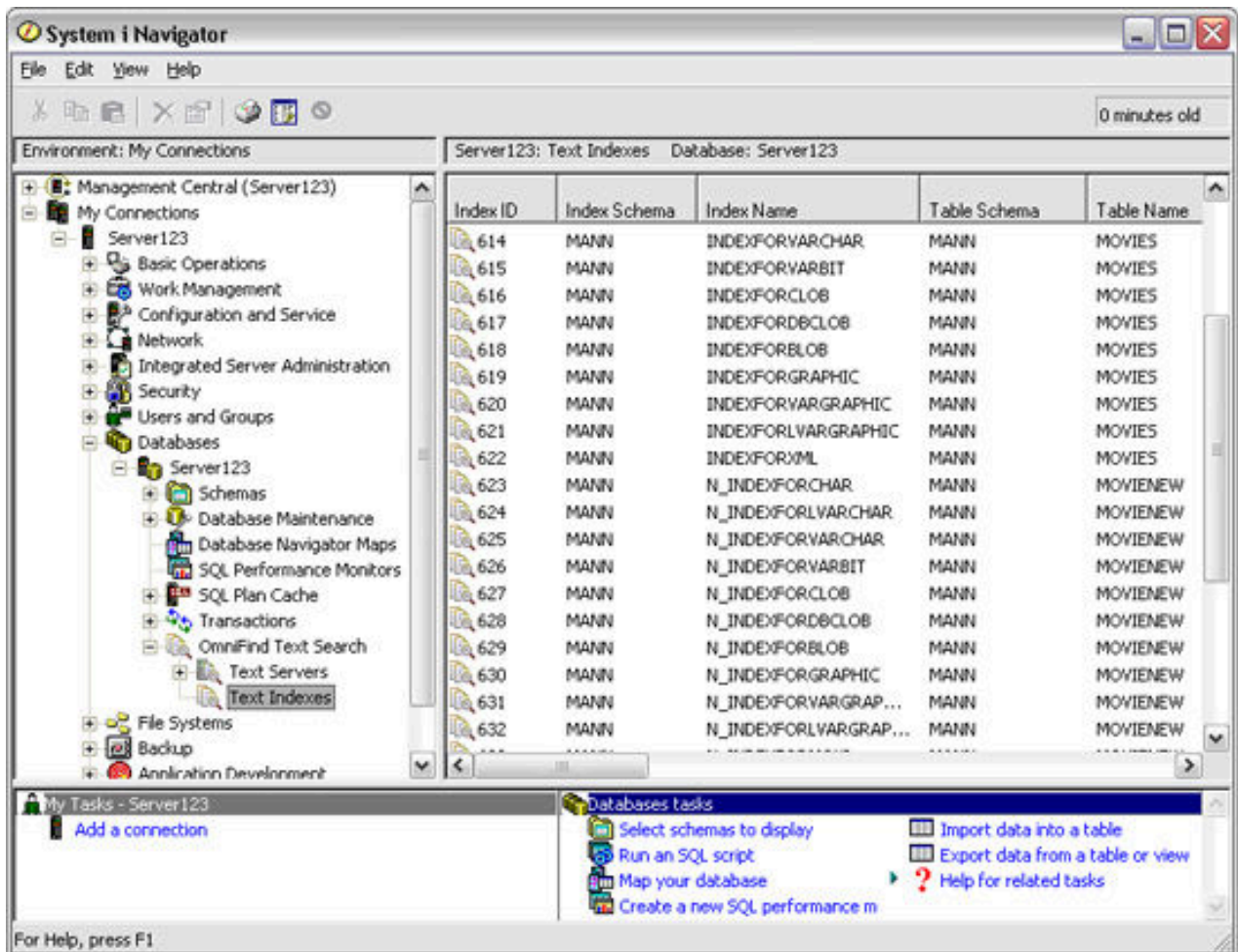
Viewing the OmniFind text indexes for a text server:

1. In the **System i Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Select **OmniFind Text Search**.
5. Select **Text Servers**.
6. Select the text server that you want to work with. The indexes for that server are displayed in the right pane.



Viewing the OmniFind text indexes for this system:

1. Select **OmniFind Text Search**.
2. Select **Text Indexes**. The indexes shown are all the text search indexes for this system in the current partition.

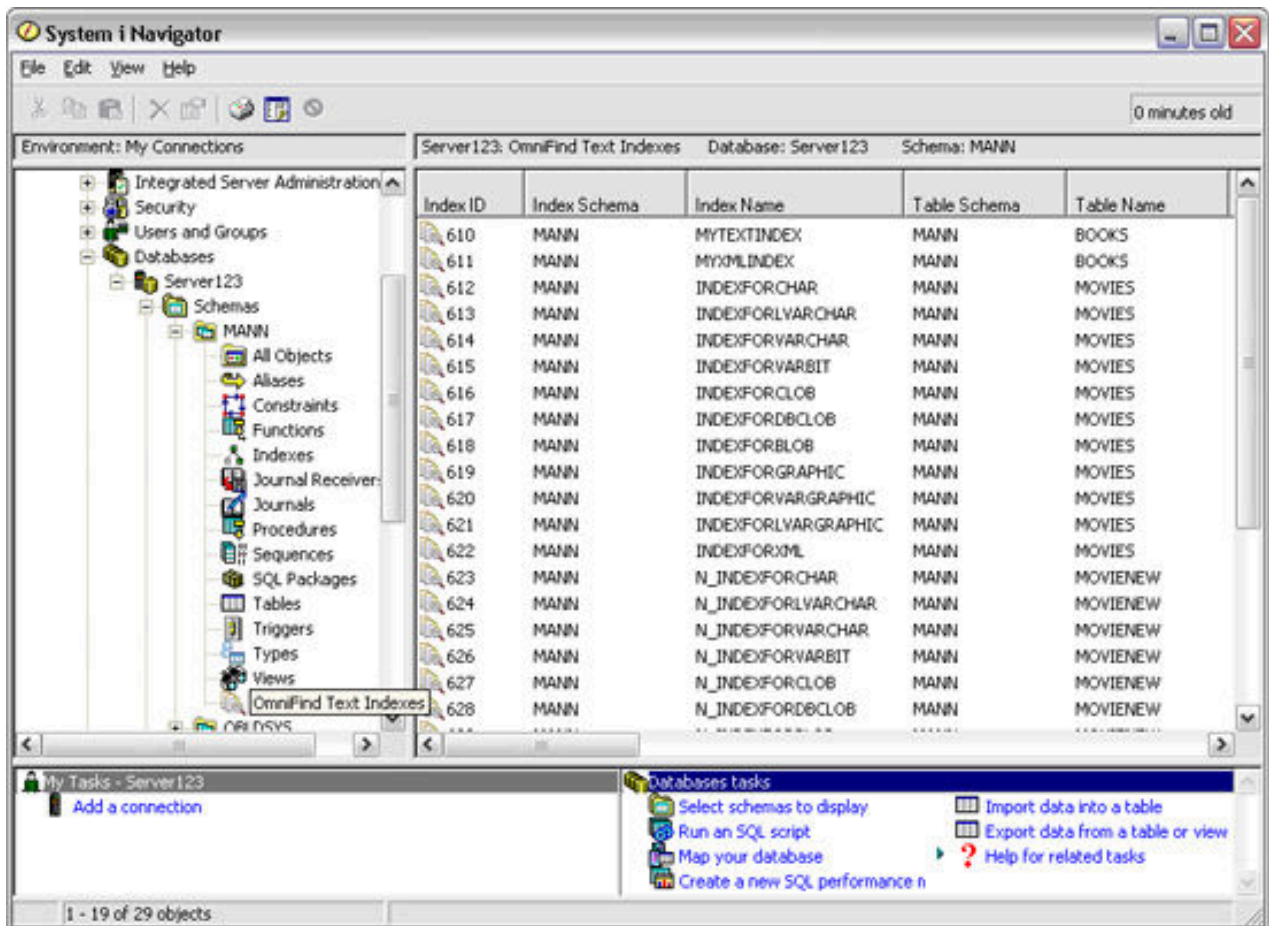


Perform operations on a text index:

1. Select an index in the right pane that you want to work with.
2. Right-click on the index.
3. Select an operation:
 - update
 - reprime
 - delete
 - view description
 - view definition
 - alter definition

Viewing the OmniFind text indexes for a schema:

1. In the **System i Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Expand **Schemas**.
5. Expand the schema that you want to work with.
6. Select **OmniFind Text Indexes**. These indexes are all the text search indexes for this schema.



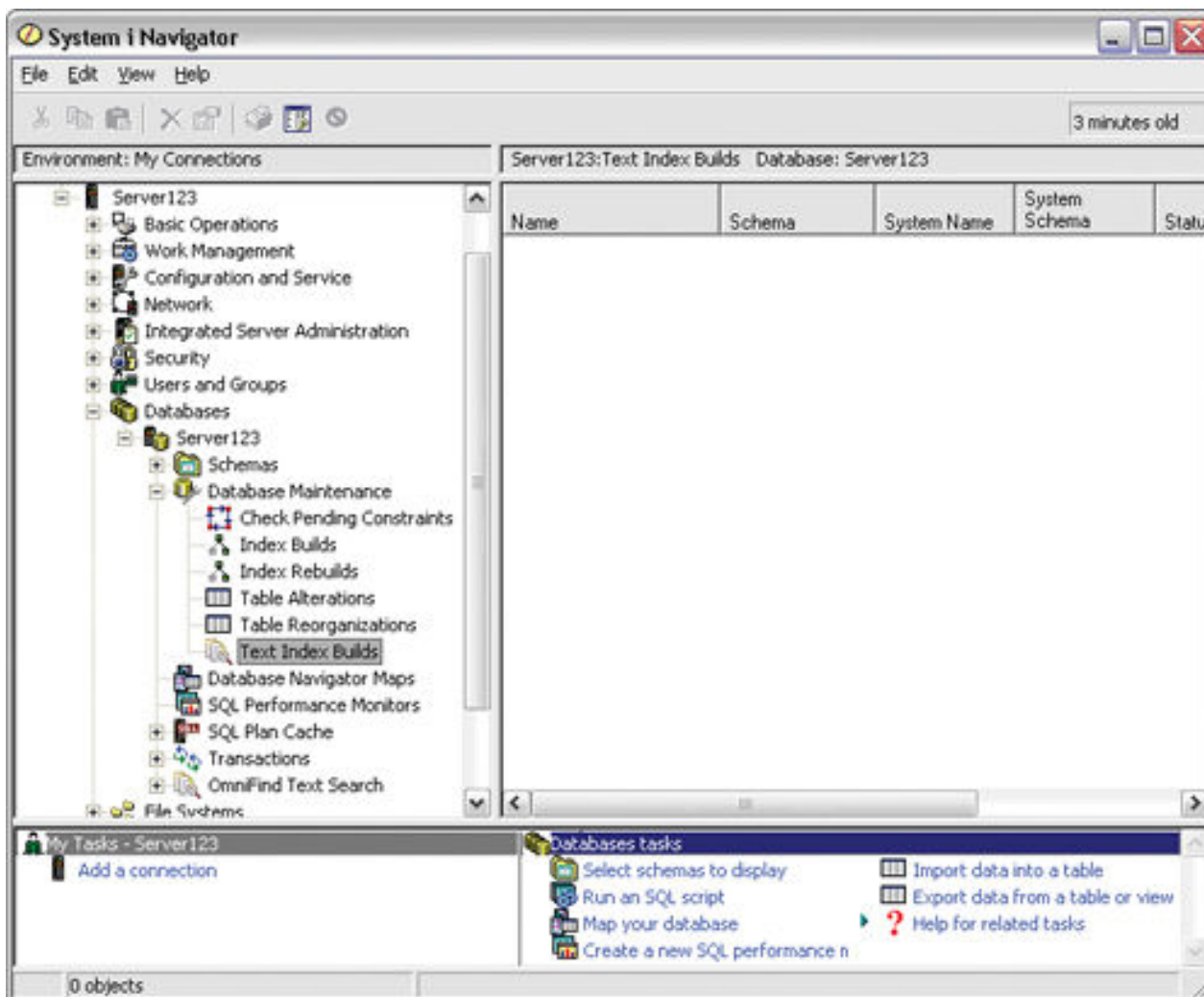
View text search index builds

You can view text indexes that are being built by the database with System i Navigator. This view is helpful in determining when text search indexes become available to your applications.

To display text search indexes that are being built, follow these steps:

1. In the **System i Navigator** window, expand the system you want to use.
2. Expand **Databases**.
3. Expand the database that you want to work with.
4. Expand **Database Maintenance**.
5. Select **Text Index Builds**.

This panel shows only text search index updates in progress. It is empty if there are no updates or repriming currently running on the system.



Text search administration tables

You can support your text search servers and indexes using the administration tables in QSYS2.

QSYS2.SYSTEXTDEFAULTS administration table

You can see the default parameters and values in the QSYS2.SYSTEXTDEFAULTS administration table. This table is created when you install OmniFind Text Search for DB2 for i.

The following table shows the contents of the QSYS2.SYSTEXTDEFAULTS administration table.

Table 33. Contents of the QSYS2.SYSTEXTDEFAULTS administration table

Column name	Data type	Nullable?	Description
NAME	VARCHAR(30)	No	Name of a default parameter for the database for text search.
VALUE	VARCHAR(512)	No	Value for the default parameter for text search.
TYPE	INTEGER	No	Reserved.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

The following table shows the contents of the QSYS2.SYSTEXTINDEXES administration table. The unique key for this table is the INDEXSCHEMA column with the INDEXNAME column. The primary key is the INDEXID column.

Table 34. Contents of the QSYS2.SYSTEXTINDEXES administration table

Column name	Data type	Nullable?	Description
INDEXID	INTEGER	No	Uniquely generated index ID for the text search index.
INDEXSCHEMA	VARCHAR(128)	No	Schema name for the text search index.
INDEXNAME	VARCHAR(128)	No	Unqualified name of the text search index.
TABLESCHEMA	VARCHAR(128)	No	Schema name of the base table.
TABLENAME	VARCHAR(128)	No	Unqualified name of the base table.
TABLEIASP	SMALLINT	No	Independent ASP of the base table.
COLLECTIONNAME	VARCHAR(255)	No	Name of the associated collection on the text search server.
SERVERID	INTEGER	No	The server ID for the text search index.
TAKEOVERSERVERID	INTEGER	Yes	Reserved for future use.
TAKEOVERSERVERPULSE	TIMESTAMP	Yes	Reserved for future use.
SEARCHARGS	VARBINARY(1024)	Yes	Reserved for future use.
ALIASSCHEMA	VARCHAR(128)	No	The alias for the schema of the base table that was used in the SYSPROC.SYSTS_CREATE stored procedure. If no alias is used, this value is identical to TABLESCHEMA.
ALIASNAME	VARCHAR(128)	No	The alias for the name of the base table that was used in the SYSPROC.SYSTS_CREATE stored procedure. If no alias is used, this value is identical to TABLENAME.
STAGINGTABLENAME	VARCHAR(128)	Yes	The name of the log table for the text search index.
EVENTTABLENAME	VARCHAR(128)	No	The name of the event table for the text search index.
OFINDEXTABLENAME	VARCHAR(128)	No	The name of the table for the text search index on the OmniFind Text Search Server for DB2 for i.
UPDATEMINIMUM	INTEGER	No	Minimum number of entries in the log table before an incremental update of the text search index is performed.

Table 34. Contents of the QSYS2.SYSTEXTINDEXES administration table (continued)

Column name	Data type	Nullable?	Description
UPDATEFREQUENCY	VARCHAR(512)	No	The update frequency for the text search index as specified by the SYSPROC.SYSTS_CREATE stored procedure.
UPDATEMODE	INTEGER	No	Indicates the update mode of the text search index. The integer 0 (zero) indicates the initial update of the text search index. A value of 1 indicates subsequent, incremental updates.
REORGANIZATIONMODE	INTEGER	No	Indicates the reorganization mode of the text search index.
CREATETIME	TIMESTAMP	No	The time that the text search index was created.
LASTUPDATETIME	TIMESTAMP	Yes	The time that the text search index was last updated.
LASTUPDATESTATUS	CHAR	Yes	<p>Indicates the internal status for optimizing the cleanup process after an initial or incremental update of the text search index. Typical values include:</p> <ul style="list-style-type: none"> • (Null) indicates that the index has never been updated. • 'C' indicates that an initial update was initiated. If the update mode is not incremental, then the initial update is either still in progress or did not complete. • 'N' indicates that an incremental update has successfully completed. <p>Other codes are used internally during update processing. The update process uses these codes to determine specific recovery actions that can be taken if the update fails to complete.</p>
SCHEDULERTASKID	INTEGER	Yes	Reserved for future use.
EXPRESSIONLISTS	CLOB (32 K)	Yes	Reserved for future use.
EXPRESSIONNUMBERS	VARBINARY(32)	Yes	Reserved for future use.
USEREXITFUNCTION	VARCHAR(18)	Yes	Reserved for future use.
REMARKS	VARCHAR(2000)	Yes	Remarks made in the COMMENTS option of the index-configuration-options parameter of the SYSPROC.SYSTS_CREATE stored procedure.

Table 34. Contents of the QSYS2.SYSTEXTINDEXES administration table (continued)

Column name	Data type	Nullable?	Description
TABLEMBR	VARCHAR(10)	Yes	The table member over which the text index is built. Used to track the specific member being indexed if the file is a multi-member file. If the value is null, the member is the first and only member of the table.

Related concepts

Server alias name

You can use a server alias name to assign a meaningful name to a server.

QSYS2.SYSTEXTCOLUMNS administration table

You can see information about the text columns for a text search index in the QSYS2.SYSTEXTCOLUMNS administration table. Each text search index has an index ID, text column names, and the schema name of the base table.

The following table shows the contents of the QSYS2.SYSTEXTCOLUMNS administration table. The primary key for this table is the INDEXID column with the COLUMNNAME column. The foreign key is the INDEXID column.

Table 35. Contents of the QSYS2.SYSTEXTCOLUMNS administration table

Column name	Data type	Nullable?	Description
INDEXID	INTEGER	No	Uniquely generated index ID for the text search index.
COLUMNNAME	VARCHAR(128)	No	Unqualified name of the text column.
TABLESCHEMA	VARCHAR(128)	No	Schema name of the base table.
TABLENAME	VARCHAR(128)	No	Unqualified name of the base table.
LANGUAGE	VARCHAR(5)	No	The language that the text search server uses for the linguistic processing of text documents. The default value is en_US (English).
FUNCTIONSCHEMA	VARCHAR(128)	Yes	The schema of a user-defined function used by OMNIFIND to access text documents that are in a column that is not of a supported data type, or that are stored elsewhere.
FUNCTIONNAME	VARCHAR(18)	Yes	The name of a user-defined function used by OMNIFIND to access text documents that are in a column that is not of a supported data type, or that are stored elsewhere.
CCSID	INTEGER	No	The coded character set identifier that is used for a text search index on a column with a binary data type.
FORMAT	VARCHAR(30)	No	The format of text documents in the column. The supported format values are TEXT, HTML, XML, and INSO.

Table 35. Contents of the QSYS2.SYSTEXTCOLUMNS administration table (continued)

Column name	Data type	Nullable?	Description
KEYCOLUMNCOUNT	INTEGER	No	The count of key columns for the text search index.
KEYCOLUMNNAMES	VARCHAR(1200)	No	The key column names for the text search index.

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

The following table shows the contents of the QSYS2.SYSTEXTSERVERS administration table. The unique key for this table is the SERVERNAME column with the SERVERPORT column. The primary key is the SERVERID column.

Table 36. Contents of the QSYS2.SYSTEXTSERVERS administration table

Column name	Data type	Nullable?	Description
SERVERID	INTEGER	No	Uniquely generated ID for the text search server.
SERVERNAME	VARCHAR(128)	No	The host name or IP address of the text search server.
SERVERADRINFO	VARBINARY(3000)	Yes	The internal representation of the SERVERNAME and SERVERPORT as determined by the SYSPROC.SYSTS_START stored procedure.
SERVERPORT	INTEGER	No	The port number for the text search server.
SERVERPATH	VARCHAR(512)	No	The server path for the text search server.
SERVERTYPE	INTEGER	No	The server type for the text search server. The value 0 (zero) indicates an IBM i text search server. The value 1 indicates a Linux text search server. The value 2 indicates a Windows text search server.
SERVERAUTHTOKEN	VARCHAR(256)	No	The authentication token for the text search server.
SERVERMASTERKEY	VARCHAR(36)	No	The server key for the text search server.
SERVERCLASS	INTEGER	No	The server class for the text search server. The value 0 (zero) indicates a production server, available for automatic selection. the value 9 indicates a test server, never allocated automatically.
SERVERSTATUS	INTEGER	No	Indicates whether the server can be used as a text search server to create new text search indexes. The default value is 0 (zero), which means that the server can be used.

Table 36. Contents of the QSYS2.SYSTEXTSERVERS administration table (continued)

Column name	Data type	Nullable?	Description
ALIASNAME	VARCHAR(128)	Yes	Alias name is unique when not null and is case sensitive.

Related concepts

Server alias name

You can use a server alias name to assign a meaningful name to a server.

QSYS2.SYSTEXTCONFIGURATION administration table

You can see the configuration parameters for the text search index, as passed by the SYSPROC.SYSTS_CREATE stored procedure, in the QSYS2.SYSTEXTCONFIGURATION administration table.

The following table shows the contents of the QSYS2.SYSTEXTCONFIGURATION administration table. The primary key is the INDEXID column with the PARAMETER column. The foreign key is the INDEXID column.

Table 37. Contents of the QSYS2.SYSTEXTCONFIGURATION administration table

Column name	Data type	Nullable?	Description
INDEXID	INTEGER	No	Uniquely generated index ID for the text search index.
PARAMETER	VARCHAR(30)	No	Parameters that are specified for the text search index in the SYSPROC.SYSTS_CREATE stored procedure.
VALUE	VARCHAR(512)	No	Values for the specified parameters.

Related reference

SYSPROC.SYSTS_CREATE

You can call the SYSPROC.SYSTS_CREATE stored procedure to create a text search index. This stored procedure enables a text column for text search indexing. The text search index can then be used in SQL queries that contain the CONTAINS or SCORE functions.

QSYS2.SYSTEXTSERVERHISTORY administration table

You can see the history of servers used for the SYSPROC.SYSTS_DROP stored procedure by viewing the auxiliary table QSYS2.SYSTEXTSERVERHISTORY.

The following table shows the contents of the QSYS2.SYSTEXTSERVERHISTORY administration table. The unique key for this table is the INDEXID column with the SERVERID column. The foreign key is the INDEXID column.

Table 38. Contents of the QSYS2.SYSTEXTSERVERHISTORY administration table

Column name	Data type	Nullable?	Description
INDEXID	INTEGER	No	The index ID for a created text search index.
SERVERID	INTEGER	No	The server ID where a text search index needs to be dropped on SYSPROC.SYSTS_DROP.

Related reference

SYSPROC.SYSTS_DROP

You can call the SYSPROC.SYSTS_DROP stored procedure to drop a text search index that was defined by using the SYSPROC.SYSTS_CREATE stored procedure.

Text Search Index view

When a text search index is created with SYSTS_CREATE, a view representing the index will be created. Querying the text search index's view can help the user obtain the status of the index. The text search index view's name is same name as the text search index's name that was specified while creating the index with SYSTS_CREATE.

The following table shows the content of text search index's view.

Column name	Data type	CCSID	Nullable?	Description
TABLESCHEMA	VARCHAR(128)	1208	No	Schema name of the base table.
TABLENAME	VARCHAR(128)	1208	No	Unqualified name of the base table.
COLUMNNAME	VARCHAR(128)	1208	No	Unqualified name of the text column from the base table.
SERVERID	INTEGER		No	Unique server ID from SYSTEXTSERVERS.
SERVERNAME	VARCHAR(128)	1208	No	Unqualified name of the text search server.
SERVERSTATUS	VARCHAR(32)	1208	No	<ul style="list-style-type: none"> 'STARTED' – Server is running. 'STOPPED' – Server is stopped.
STAGINGTABLENAME	VARCHAR(128)	1208	No	Unqualified name of the log table for the text search index.
LASTUPDATETIME	TIMESTAMP		Yes	The time that the text search index was last updated.
LASTUPDATESTATUS	VARCHAR(30)	1208	Yes	<ul style="list-style-type: none"> 'NEVER UPDATED' – The index was not updated before. 'UP TO DATE' – Last update succeed and there is no pending changes to do. 'CHANGES PENDING' – Last update succeed, but there are still new changes to be updated. 'FAILED' – Last update failed.
UPDATEMINIMUM	INTEGER		No	Minimum number of entries in the log table before an incremental update of the text search index is performed.

Table 39. Contents of the view created by SYSTS_CREATE (continued)

Column name	Data type	CCSID	Nullable?	Description
UPDATEFREQUENCY	VARCHAR(512)	1208	No	The update frequency for the text search index as specified by the SYSPROC.SYSTS_CREATE stored procedure.
PENDINGCOUNT	INTEGER		No	Indicate how many rows to be indexed in next update process.

Extensions to Index and Search Non-DB2 Data

The OmniFind Text Search for DB2 for i provides an additional set of stored procedures to create, administer, and search text search collections.

Extensions Overview

OmniFind Text Search Server for DB2 for i provides an additional set of stored procedures to create, administer, and search text search collections. A text search collection is used to index data associated with system objects such as spool files in an output queue, or stream file data in the integrated file system.

A text search collection describes one or more sets of system objects that will have their associated text data indexed and searched. For example, a collection may contain an object set of all spool files in output queue QUSRSYS/QEZJOBLOG, and/or an object set for all stream files in directory '/home/alice/text_data'.

The text search collection referred to in this documentation should not be confused with a DB2 schema (sometimes also referred to as a collection), or a Lucene collection (Part of the internal structure of a DB2 text search index).

When a text search collection is created, several DB2 objects are created on the system in an SQL schema. The following objects will be created in the schema:

- Catalogs for tracking the collection's configuration.
- Catalogs for tracking the objects that have been indexed.
- SQL Stored procedures to administer and search the collection.
- A DB2 text search index for indexing the associated text.

Administration of the collection is provided with stored procedures, most of which are created in the schema.

An explanation of other OmniFind Text Search Server for DB2 for i enhancements can be found at: [OmniFind Text Search Server for DB2 for i enhancements](#).

Creating a Text Search Collection

This procedure creates an empty search collection. An SQL schema will be created on the system to contain information about the collection. The schema will contain DB2 objects necessary to track and index objects.

SYSPROC.SYSTS_CRTCOL and SYSPROC.SYSTS_CREATE_COLLECTION

Authorization

SYSTS_CRTCOL and SYSTS_CREATE_COLLECTION will be shipped with *EXECUTE authority granted to public.

These procedures will not adopt any additional authority and will run under the invoking profile.

In order to create a text search collection the invoker must have:

- Authority to create a DB2 schema
- Authority/ability to create a text search index

The DB2 Objects created as part of the collection, including the administrative stored procedures are created with public authority *EXCLUDE

The user profile creating the collection owns all objects in the collection. A user may grant authority to specific procedure to other users in order to allow another user to administer and search the text search collection.

Syntax

```
► SYSTS_CREATE ( ( indexSchema , indexName , textSource ,  
                  null )  
                  ► options ) ►
```

Syntax

```
► SYSTS_CREATE_COLLECTION ( collection_name )  
                          , options  
                          , asp_device_name ) ►
```

The schema qualifier is SYSPROC.

Parameters

collection_name

Name of Collection. This name uniquely identifies the collection and must be a non null string. A schema of the name specified for collection name will be created to hold the associated DB2 objects.

Note: Enclose names in double quotation marks if the names conflict with SQL keywords or OmniFind keywords that can be used.

The collection name parameter will follow the SQL rules for schema names.

The collection name must not match the name of an existing user profile.

The data type of this parameter is VARCHAR(128).

options

A character string that specifies the various options that are available for this stored procedure.

The data type for this parameter is VARCHAR(32000).

```
options  
┌──────────────────┬──────────────────┬──────────────────┐  
│ text-default-information │ update-characteristics │  
└──────────────────┴──────────────────┴──────────────────┘  
┌──────────────────┬──────────────────┐  
│ index-configuration-options │  
└──────────────────┴──────────────────┘
```

text-default-information



text-default-information

Specifies the language that is used when processing documents, and the format of text documents in the column.

LANGUAGE *language*

Specifies the language that OmniFind Text Search Server for DB2 for i uses for the linguistic processing of text documents. The default value is en_US (English). If you specify the value for this parameter as AUTO, OmniFind Text Search Server for DB2 for i tries to determine the language.

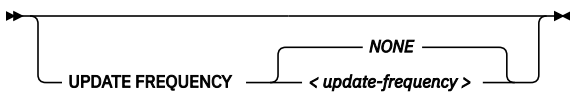
Important: If the language of the documents is not English, do not use the default value of en_US. Change the value to the language of the documents; otherwise, linguistic processing does not work as expected.

FORMAT *format*

Identifies the format of text documents to be indexes, such as TEXT or INSO. The OmniFind Text Search Server for DB2 for i needs to know the format, or content type, of the text documents that you intend to index and search. If you do not specify the *format* parameter, the default is TEXT.

The *format* value INSO allows OmniFind Text Search Server for DB2 for i to determine the format. If the OmniFind Text Search Server for DB2 for i cannot determine the document format, then a document error is logged in the job log during processing by the UPDATE stored procedure.

update-characteristics



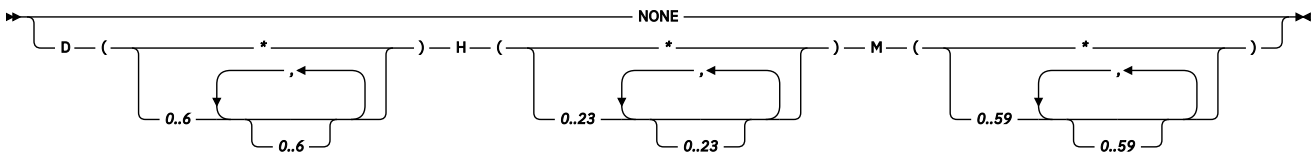
update-characteristics

Specifies the frequency of automatic updates to the text search collection. The update process for a text search collection involves both indexing the text data, and crawling system objects to detect new or changed data.

UPDATE FREQUENCY *update-frequency*

Specifies when to make automatic updates to the text search collection. The default value is NONE.

update-frequency (Format 1)



NONE

If NONE is specified, then no further index updates are made. The update must be started manually.

D

Specifies the day or days of the week when the index is updated. An asterisk (*) specifies all days. 0 specifies Sunday.

H

Specifies the hour or hours when the index is updated. An asterisk (*) specifies all hours.

M

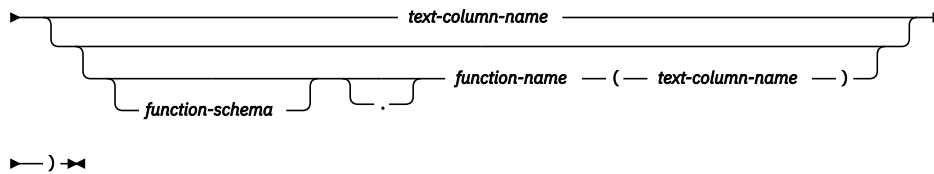
Specifies the minute or minutes when the index is updated. An asterisk (*) cannot be specified. The minimum update frequency is 5 minutes.

Example: This example specifies that the index update is to run every 30 minutes.

```
UPDATE FREQUENCY D(*) H(*) M(0,30)
```

update-frequency (Format 2, chronological)

► *tableSchema* . *tableName* — (→



The format of the *update-frequency (chronological)* option is a list of the five values separated by a blank space. The five values represent the minutes, hours, days of the month, months of the year, and days of the week beginning with Sunday.

If you specify an interval of values or an asterisk (*), you can specify a step value by using a forward slash (/) at the end of the defined interval.

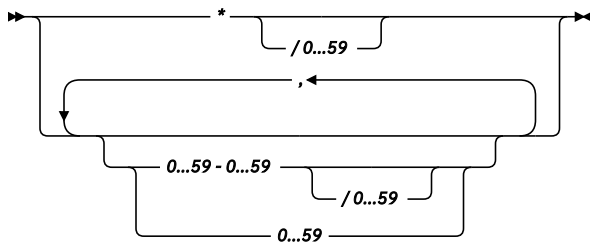
Example: This example specifies that the index update is to run every quarter hour (0, 15, 30, 45) on the even hours between 8 a.m. and 6:45 p.m. (8-18/2 is equivalent to 8, 10, 12, 14, 16, 18), from Monday to Friday every month of the year (* * 1-5).

```
0,15,30,45 8-18/2 * * 1-5
```

minute

Specifies the minutes of the hour when the text search index is to be updated. You can specify an asterisk (*) for an interval of every 5 minutes, or you can specify an integer 0 - 59. You cannot repeat values. The minimum update frequency is 5 minutes. A value of 1,4,8 is not valid.

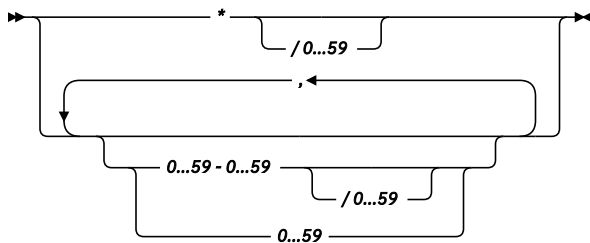
update-frequency (minute)



hour

Specifies the hours of the day when the text search index is to be updated. You can specify an asterisk (*) for every hour, or you can specify an integer 0 - 23. You cannot repeat values.

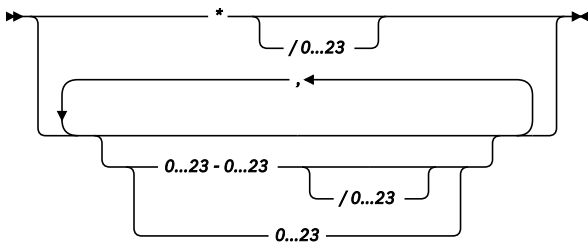
update-frequency (hour)



dayOfMonth

Specifies the days of the month when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 1 - 31. You cannot repeat values.

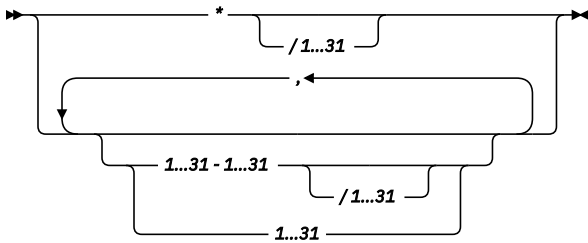
update-frequency (dayOfMonth)



monthOfYear

Specifies the months of the year when the text search index is to be updated. You can specify an asterisk (*) for every month, or you can specify an integer 1 - 12. You cannot repeat values.

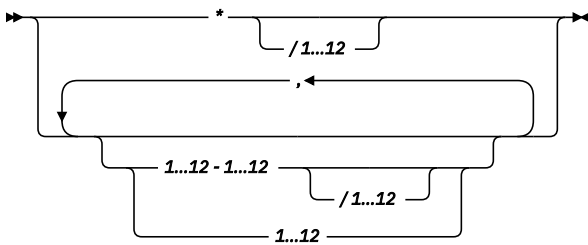
update-frequency (monthOfYear)



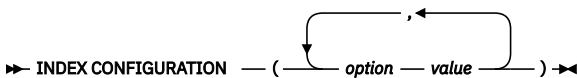
dayOfWeek

Specifies the days of the week when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 0 - 7. Both 0 and 7 are valid values for Sunday. You cannot repeat values.

update-frequency (dayOfWeek)



index-configuration-options



index-configuration-options

Specifies additional index-specific values as option value pairs. You must enclose string values in single quotation marks. A single quotation mark within a string value must be represented by two consecutive single quotation marks.

CJKSEGMENTATION

Specifies the segmentation method to use when you index documents for CJK (Chinese, Japanese, Korean) languages. The supported values are MORPHOLOGICAL and NGRAM. If the CJKSEGMENTATION value is not specified, the default value is used. The default value is specified by the DEFAULTNAME value in the QSYS2.SYSTEXTDEFAULTS table.

SERVER

Specifies the ID or alias name of the server to be used to store the text search index. If an ID is used, the value is an integer that must exist in the SERVERID column of the QSYS2.SYSTEXTSERVERS catalog. If an alias name is used, the value is a string that must exist in the ALIASNAME column of the QSYS2.SYSTEXTSERVERS catalog. If SERVER is not specified, the default is to select the server with the fewest text search indexes from the servers in

the QSYS2.SYSTEXTSERVERS table where parameter SERVERSTATUS is set to 0 (zero), which means that the server is available.

UPDATEAUTOCOMMIT

Specifies how often a commit operation is performed when fetching documents during an index update. A value of 0 (zero) means that a commit operation occurs only at the end of processing.

The value must be an integer between 0 (zero) and 2147483647. The default value is 100.

Performance tip: The value of UPDATEAUTOCOMMIT can have a substantial impact on the performance of index updates. The commit operation that takes place at the specified interval ensures a consistent checkpoint from which to restart the index update, if it is interrupted. However, the commit also temporarily suspends the update process. Increasing the UPDATEAUTOCOMMIT value (or setting it to 0) can substantially improve the update performance, especially the initial update. The value you specify must balance the need for performance with the need for recoverability, based on the frequency of the index updates.

asp_device_name

This parameter, if specified and not null, determines which Auxiliary storage pool the collection is created into. This parameter is optionally available to match the CREATE SCHEMA capability to create the DB2 objects on a specific ASP device.

If a value is supplied for this parameter, the value must name a disk pool in the primary asp group of the current namespace, or a basic ASP unit if the namespace is the system ASP only.

The data type for this parameter is VARCHAR(10).

Examples

1. CALL SYSTS_CRTCOL('mycollection');

A collection MYCOLLECTION is created.

2. CALL SYSTS_CRTCOL("mycollection", "UPDATE FREQUENCY D(*) H(*) M(0)")

A collection mycollection (lower case not including the delimiters) is created.

The text search collection will have an update frequency of every day, at the top of every hour.

3. CALL SYSTS_CRTCOL('ur_collection','UPDATE FREQUENCY NONE ' || ' LANGUAGE zh_CN INDEX CONFIGURATION(' || ' CJKSEGMENTATION MORPHOLOGICAL) ' , '23')

A collection UR_COLLECTION is created.

The collection has no update frequency.

The collection's language is simplified Chinese, using dictionary linguistic analysis (morphological).

The SQL schema is created in basic ASP #23.

When the text search collection is created, procedures are created in the DB2 schema to administer the collection.

To create a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > Omnifind > Create Collection**.

Adding an Object Set for Spool File Data

The stored procedure is in the DB2 schema to add an object set for spool file data.

Note: only SNA Character String (SCS) data is supported. Spool files that contain other types of data cannot be indexed and will result in a document level error when encountered. The error will be logged

in the job log, and indexing will continue with the next spool file. The GET_OBJECTS_NOT_INDEXED procedure can be used to determine which spool files were not included in the indexing process.

ADD_SPLF_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection. will be shipped with *EXECUTE authority granted to public.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

Add a Spool File Object Set:

```

▶ — ADD_SPLF_OBJECT_SET — ( output_queue_lib , →
                             null
▶ output_queue_name , user_name , qual_job_name , →
   null null null
▶ qual_job_user , qual_job_number , user_data , →
   null null null
▶ begin_timestamp , ending_timestamp , — output_set_id ) →
   null null ,

```

The following simplified versions of the add spool file object set procedure may be used for convenience.

Select spool files by output queue only:

```

▶ ADD_SPLF_OBJECT_SET — ( output_queue_lib , output_queue_name ) →
                             null null

```

Select spool files by output queue, and user name only:

```

▶ ADD_SPLF_OBJECT_SET — ( output_queue_lib , output_queue_name , →
                             null null
▶ user_name ) →
   null

```

The schema qualifier is the name of the text search collection.

Parameters

output_queue_lib

A system name for the output queue library.

A NULL value or empty string indicates that output queues in all libraries will be included in the index.

The data type for this parameter is VARCHAR(10)

[See system name conventions]

[See additional restrictions]

output_queue_name

A system name for the output queue name.

A NULL value or empty string indicates that any output queue will be included in the index.

The data type for this parameter is VARCHAR(10)

[See system name conventions]

[See additional restrictions]

user_name

A system name for the user that owns the spool file.

A NULL value or empty string indicates that no filtering on the user name will be performed.

The data type for this parameter is VARCHAR(10)

[See system name conventions]

[See additional restrictions]

qual_job_name

Name of the job that created the spool file.

A NULL value or empty string indicates that no filtering on the job name will occur.

The data type for this parameter is VARCHAR(10)

[See system name conventions]

qual_job_user

System name for the user profile name of the job associated with the spool file.

A NULL value or empty string will indicate that no filtering on the job user will occur.

The data type for this parameter is VARCHAR(10)

A non-null, non-empty value for this parameter is required if QUALIFIED_JOB_NAME is specified.

If QUALIFIED_JOB_NAME is empty string or NULL, this parameter must also be NULL or empty string.

[See system name conventions]

[See additional restrictions]

qual_job_number

A six character string representing the job number. (Must be digits 0-9)

A NULL value or empty string will indicate that no filtering on the job number will occur.

The data type for this parameter is VARCHAR(6)

A non-null, non-empty value for this parameter is required if QUALIFIED_JOB_NAME is specified.

If QUALIFIED_JOB_NAME is empty string or NULL, this parameter must also be NULL or empty string.

[See system name conventions]

[See additional restrictions]

user_data

A ten character string that the user associates with a spool file.

This string is not converted to uppercase, and must exactly match the user data associated with a spool file in order to be considered a match.

Note: A value of 'abc' is different than 'ABC'

A NULL value, or empty string will indicate that no filtering on the user data will occur.

The data type for this parameter is VARCHAR(10)

begin_timestamp

Timestamp value indicating the earliest creation time that will be included in the collection. Spool files created earlier than this timestamp will not be indexed.

A value of NULL can be provided to indicate that any spool file created before the ending creation timestamp value should be indexed. If END_TIMESTAMP is also NULL, then no filtering on the creation timestamp will occur.

The data type for this parameter is TIMESTAMP

end_timestamp

Timestamp value indicating the latest creation time that will be included in the collection. Spool files created after this timestamp will not be indexed.

A value of NULL can be provided to indicate that any spool file created after the BEGIN_TIMESTAMP will be indexed. If BEGIN_TIMESTAMP is also NULL, then no filtering on the creation timestamp will occur.

The data type for this parameter is TIMESTAMP

output_set_id

Output Integer value that returns the set id for the object set that was added.

This value can be used to remove the object set at a later time.

This parameter is optional.

The data type for this parameter is INTEGER

System Naming Conventions

Parameters that require system names as input must be valid system names or an error will occur. This parameters will be processed the same way the command analyzer processes names for CL commands. See [Object naming rules](#) for more information on system names

```
call nick12345.add_splf_object_set('ntl', 'justtext', 'ntl', "", "", "", NULL, NULL);
```

The filter information passed to the procedure will be an output queue NTL/JUSTTEXT for user NTL (converted to uppercase)

Unlike SQL names, for a system name the delimiters will remain on the name, but only if necessary. call `nick12345.add_splf_object_set('ntl', 'justtext', 'NTL', "", "", "", NULL, NULL);`

The filter information passed to the api will be output queue "ntl"/"justtext" for user NTL (no quotes around user NTL)

Note: The stored procedure has a ten character limit on the interface and does not support unnecessary double quotes that cause this limit to be exceeded.

Invalid system names will cause an error.

Additional Restrictions

- Generic names are not supported. In other words it is not possible to index all output queues that start with MYOUT by adding an object set for MYOUT*
- The output queue library name and output queue name must either be both null (or empty string)...or both have valid system names. It is not possible filter on all output queues in library xyz, or to filter on all output queues named 'abc' in any library.
- If a specific output queue name and library are specified, that output queue must exist at the time the object set is added. There is no logic to prevent the deletion of the output queue at some later time, but the object set will effectively become 'empty'.
- If both USER_NAME and QUALIFIED_JOB_USER are non-null, non-empty values, then they must be equal. A spool file owner will always match the qualified job user, and therefore these values can never be different when both are used as a filter.

Authorities to Indexed Objects

When adding a spool file object set, consider the authority requirements needed to retrieve the text from the spool files carefully. These authorities will be a factor when the UPDATE stored procedure is invoked. See the update stored procedure documentation for information on the authority requirements to indexed objects.

Examples

Add an object set to collection nick789 for all spool files in output queue NTL/MYOUTQ.

```
> call nick789.add_splf_object_set('NTL', 'MYOUTQ');
```

Add an object set to collection nick123 to index all spool files owned by user NTL.

```
> call nick123.add_splf_object_set('', '', 'NTL');
```

Add an object set to collection default_search_col to index all spool files created in 2010

```
> call default_search_col.add_splf_object_set('', '', '', '', '', '', '', '2010-01-01T00:00:00', '2011-01-01T00:00:00');
```

Add an object set to collection default_search_col to index all spool files created in 2010 with user data 'MYAPP':

```
> call default_search_col.add_splf_object_set('', '', '', '', '', '', 'MYAPP', '2010-01-01T00:00:00', '2011-01-01T00:00:00');
```

To add spool file object set to a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management** > **System** > **All Tasks**.
2. On the right panel, select **System** > **OmniFind** > **Collection List**.
3. Right click the collection and select **Properties**. On the **Object** tab, press **Add Output Queues** or **Add Spooled Files** button.

Removing Object Set for Spool File Data

This stored procedure will remove an object set for spool file data from a text search collection.

RMV_SPLF_OBJECT_SET

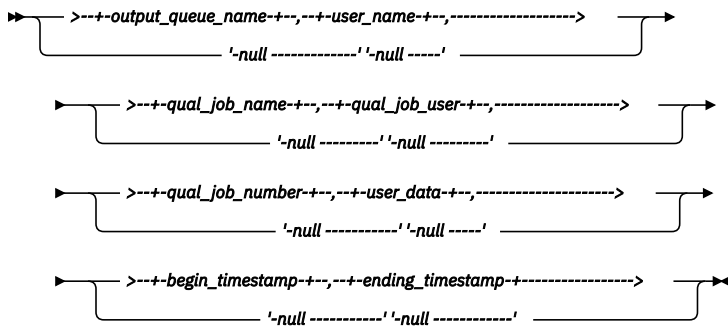
Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

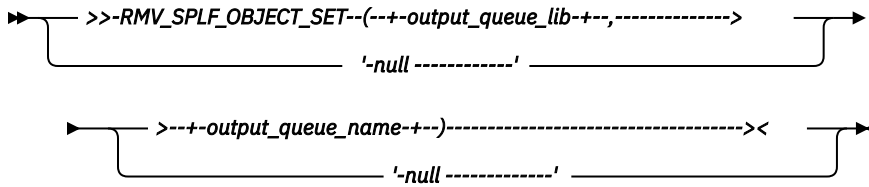
Syntax

Remove a Spool File Object Set:

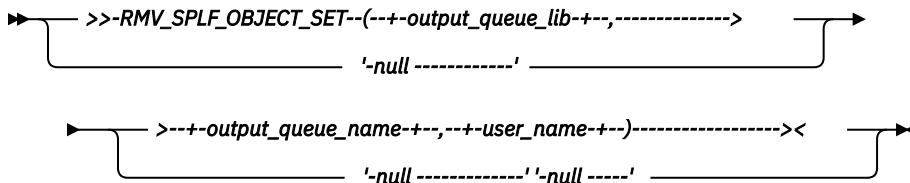


The following simplified versions of the remove spool file object set procedure may be used for convenience.

Select spool files by output queue only:



Select spool files by output queue, and user name only:



The schema qualifier is the name of the text search collection.

Parameters

output_queue_lib

A system name for the output queue library.

The data type for this parameter is VARCHAR(10)

output_queue_name

A system name for the output queue name.

The data type for this parameter is VARCHAR(10)

user_name

A system name for the user that owns the spool file.

The data type for this parameter is VARCHAR(10)

qual_job_name

Name of the job that created the spool file.

The data type for this parameter is VARCHAR(10)

qual_job_user

System name for the user profile name of the job associated with the spool file.

The data type for this parameter is VARCHAR(10)

A non-null, non-empty value for this parameter is required if QUALIFIED_JOB_NAME is specified.

If QUALIFIED_JOB_NAME is empty string or NULL, this parameter must also be NULL or empty string.

qual_job_number

A six character string representing the job number. (Must be digits 0-9)

The data type for this parameter is VARCHAR(6)

A non-null, non-empty value for this parameter is required if QUALIFIED_JOB_NAME is specified.

If QUALIFIED_JOB_NAME is empty string or NULL, this parameter must also be NULL or empty string.

user_data

A ten character string that the user associates with a spool file.

This string is not converted to uppercase, and must exactly match the user data associated with a spool file in order to be considered a match.

Note: A value of 'abc' is different than 'ABC'

The data type for this parameter is VARCHAR(10)

begin_timestamp

This timestamp value indicating the earliest creation time of the spool files added in the object set.

The data type for this parameter is TIMESTAMP

end_timestamp

This timestamp value indicating the latest creation time of the spool files added in the object set.

The data type for this parameter is TIMESTAMP

Note: For more detail meaning and restrictions of above parameters Please refer store procedure add_splf_object_set description.

Result Note

In following case remove will be failed due to object set not found:

>User inputs incorrect parameters

>Specifies parameters corresponding object set has been deleted previously User will get error message show that the object set doesn't exist with the specific attribute.

QUERY_OBJECT_SET() returns the object set list and the input parameters. User can specify correct input parameters while invoking this stored procedure to remove the object set.

Examples

Remove an object set in collection nick789 for all spool files in output queue NTL/MYOUTQ.

```
> call nick789.rmv_splf_object_set('NTL', 'MYOUTQ');
```

Remove an object set in collection nick123 for all spool files owned by user NTL.

```
> call nick123.rmv_splf_object_set('', '', 'NTL');
```

Remove an object set in collection test_col for all spool files created in 2010

```
> call test_col.rmv_splf_object_set('', '', '', '', '', '', '', '2010-01-01T00:00:00',  
'2011-01-01T00:00:00');
```

Add an object set to collection default_search_col to index all spool files created in 2010 with user data 'MYAPP':

```
> call default_search_col.add_splf_object_set('', '', '', '', '', '', 'MYAPP',  
'2010-01-01T00:00:00', '2011-01-01T00:00:00');
```

Adding an Object Set for Stream File Data

The stored procedure is in the DB2 schema to add an object set for stream file data.

ADD_IFS_STMF_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

This procedure allows a user to add an object set of stream files (STMF) in the Integrated File System (IFS).

Add an object set for stream file data (stream files in IFS):

```
► ADD_IFS_STMF_OBJECT_SET ( — stmf_expression_string — ) ►  
                        , — output_set_id —
```

The schema qualifier is the name of the text search collection.

Parameters

stmf_expression_string

This parameter contains an absolute path to a directory containing the files that will be indexed.

This must be a valid directory (type *DIR) on a file system that is accessible. Stream file objects (type *STMF) within this directory will be indexed. The path name should be absolute and should not contain any regular expressions.

The data type for this parameter is VARCHAR(32000)

Stream files contained within the specified directory are indexed.

- Symbolic links are NOT followed
- Sub-directories are NOT processed
- Path names must not be delimited, characters such as '*', '?', etc do not have any special meaning and should not be escaped.
- Path names may or may not be case sensitive, depending on the attribute of the file system.

A check will be performed when adding the object set to verify that a duplicate set does not already exist in the text search collection. This check does not consider equivalent paths to be duplicate.

In other words the following paths could all represent the same directory, but will be considered unique object sets, furthermore, the objects in those sets will be indexed multiple times as unique objects.

```
/dir1/DIR2  
/dir1//DIR2//  
/DIR1/DIR2/ (if file system is case insensitive)  
/dir1/DIR2/./DIR2  
etc.
```

output_set_id

Output Integer value that returns the set id for the object set that was added. This value can be used to remove the object set at a later time.

This parameter is optional.

The data type for this parameter is INTEGER.

Special Considerations for Update Processing

Non-existent file systems:

If a directory cannot be located during an update operation, the files associated with that directory will not be removed from the index. This avoids unnecessary re-indexing of documents when a files system is unmounted and later remounted.

If these files need to be removed from the index, several options exist:

- Issue the remove object set stored procedure against the IFS Stream file object set. This will remove any documents associated with the object set.
- Issue the REPRIME stored procedure against the collection. All data will be removed from the index, and only files that can be located will be reindexed.
- Create the directory as an empty directory and issue the update

CCSID Conversion

If the collection's FORMAT is TEXT:

- The CCSID attribute of the file is used to convert the file's extracted data to UTF-8 for indexing. The CCSID attribute of the file must be correct in order for the file to be correctly indexed.

If the collection's FORMAT is INSO:

- The data from the file will be extracted from the file and sent to the text search server for processing. No character set conversion will occur, and the CCSID attribute of the file will be ignored. The text search server will use its rich text processing to determine the format and encoding of the document. This can be used to index rich text (such as PDF) files, or ordinary text files. For some plain text documents, it may not be possible for the text search server to determine encoding of the document with enough confidence to index the data. This is more likely for very small documents, but can occur for larger documents that use a wide range of characters as well. If the format and encoding of the file cannot be determined, the file will not be indexed and a document error will be logged.

Authorities to Indexed Objects

When adding an IFS stream file object set, consider the authority requirements to read the stream files carefully. Adopted authorities are not honored when accessing the stream file's data. In addition, scheduled updates run under the user profile that owns the index. See the update stored procedure documentation for information on the authority requirements to indexed objects.

ADD_IFS_STMF_OBJECT_SET_WITH_SUBDIR

The syntax and authority requirement of this stored procedure are similar as ADD_IFS_STMF_OBJECT_SET. With this stored procedure, user can add a directory as an object set to the collection. All the files and subdirectories under this directory will be indexed recursively.

Example

Add an object set to MYCOLLECTION to index all stream files in an IFS directory '/home/ntl/stmf':

```
> CALL MYCOLLECTION.ADD_IFS_STMF_OBJECT_SET('/home/ntl/stmf');
```

Add an object set to MYCOLLECTION to index all stream files and subdirectories in an IFS directory '/home/ntl/stmf':

```
> CALL MYCOLLECTION.ADD_IFS_STMF_OBJECT_SET_WITH_SUBDIR('/home/ntl/stmf');
```

To add IFS path to a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Properties**. On the **Object** tab, press **Add IFS Path** button. Choose **Include sub-directories** to add all sub directories under the specified IFS path.

Removing Object Set for Stream File Data

The stored procedure is in the DB2® schema to remove an object set for stream file data.

RMV_IFS_STMF_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

This procedure allows a user to remove an object set of stream files (STMF) in the Integrated File System (IFS).

Remove an object set for stream file data (stream files in IFS):

```
► >>-RMV_IFS_STMF_OBJECT_SET (--stmf_expression_string-----><   ✎
```

The schema qualifier is the name of the text search collection.

Parameters

stmf_expression_string

This parameter is an absolute path to a directory which is the object set attribute.

It's no need to be a valid directory on a file system that is accessible since the path could be deleted by user, but object set still exist. The path name should be absolute and should not contain any regular expressions.

The data type for this parameter is VARCHAR(32000)

Note: Note for the specified directory path name:

- Path names must not be delimited, characters such as '*', '?', etc do not have any special meaning and should not be escaped.
- Path names may or may not be case sensitive, depending on the attribute of the file system.

Since RMV_IFS_STMF_OBJECT_SET does not consider equivalent paths to be duplicate object set, user should indicate exactly the same path as ADD_IFS_STMF_OBJECT_SET added when try to remove the specific object set.

In other words the following paths could all represent the same directory, but will be considered as different object sets.

```
/dir1/DIR2  
/dir1//DIR2//  
/DIR1/DIR2/ (if file system is case insensitive)  
/dir1/DIR2/./DIR2  
etc.
```

RMV_IFS_STMF_OBJECT_SET_WITH_SUBDIR

This stored procedure can remove an object set which include sub directory IFS files. Such object set could be added by procedure

ADD_IFS_STMF_OBJECT_SET_WITH_SUBDIR.

The syntax, authority requirement and parameters of this stored procedure are similar as RMV_IFS_STMF_OBJECT_SET. If user add an IFS path to both with sub-dir object set and without sub-dir object set, use this procedure will only remove the one object set with sub-dir.

Result Note

In following case remove will be failed due to object set not found:

>User inputs incorrect IFS path

>Specifies IFS path corresponding object set has been deleted previously User will get error message show that the object set doesn't exist with the specific attribute.

QUERY_OBJECT_SET() returns the object set list and the input parameters. User can specify correct input parameters while invoking this stored procedure to remove the object set.

Examples

Remove an object set in MYCOLLECTION which IFS directory is '/home/ntl/stmf':

```
> CALL MYCOLLECTION.RMV_IFS_STMF_OBJECT_SET('/home/ntl/stmf');
```

Remove an object set in MYCOLLECTION which include all stream files and subdirectories in an IFS directory '/home/ntl/stmf':

```
> CALL MYCOLLECTION.RMV_IFS_STMF_OBJECT_SET_WITH_SUBDIR('/home/ntl/stmf');
```

Adding an Object Set for Multiple Members Source Physical File

The stored procedure is in the DB2® schema to remove an object set for multi-member source physical file.

ADD_SRCPF_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

This procedure allows a user to add an object set of multi-member source physical file (MMPF).

Add an object set for multi-member source physical file:

```
>>-ADD_SRCPF_OBJECT-SET -(---SRCPF_LIB----->
>----- SRCPF_NAME -----)<-----<
>--+-----+-->-----<
'--,-- OUT_SETID -'
```

The schema qualifier is the name of the text search collection.

Parameter

SRCPF_LIB

This parameter means an absolute library which contains the source physical files that will be indexed.

This must be a valid library name in the system. Multiple members source physical file object within this library will be indexed. The library name should be absolute and should not contain any regular expressions.

The data type for this parameter is VARCHAR(10)

SRCPF_NAME

This parameter means an absolute source physical file which can have one or more members, all the members in the source physical file will be indexed. The file name should be absolute and should not contain any regular expressions.

Note: If the source physical file is deleted after adding the object set to the text search collection, the subsequent call to the UPDATE stored procedure will detect it, then the search result will not contain the members of that source physical file.

****ALLSRCPF***

If *ALLSRCPF was specified as source physical name, it means all the source physical files in SRCPF_LIB will be indexed while updating index.

Note: If user specify *ALLSRCPF as source physical name while adding an object set, it's also allowed that adding another specific PF under the same library as an object set. This will NOT lead to duplicate error.

The data type for this parameter is VARCHAR(10)

OUT_SETID

Output Integer value that returns the set id for the object set that was added. This value can be used to remove the object set at a later time.

This parameter is optional.

The data type for this parameter is INTEGER.

Additional Restrictions

- A check will be performed when adding the object set to verify that a duplicate set does not already exist in the text search collection.
- While adding a source physical file object set to a text search collection, OmniFind verifies that the source physical file exists. If the source physical file does not exist, an error message is returned and the object set is not added.
- If the file specified is not a source physical file, the procedure call fails with an error.

Authorities to Indexed Objects

When adding a Multiple members source physical file object set, consider the authority requirements to read the files carefully. Adopted authorities are not honored when accessing the source physical file's data. In addition, scheduled updates run under the user profile that owns the index. See the update stored procedure documentation for information on the authority requirements to indexed objects.

Examples

Add an object set to MYCOLLECTION to index the multi-member source physical file QCSRC in library ISVSQLLP:

```
> CALL MYCOLLECTION.ADD_SRCPF_OBJECT_SET('ISVSQLLP', 'QCSRC');
```

Add an object set to MYCOLLECTION to index the multiple members source physical file QCSRC in a library ISVSQLLP, and expect to get the setid.

```
> create variable setid int default 0;  
> CALL MYCOLLECTION.ADD_SRCPF_OBJECT_SET('ISVSQLLP','QCSRC',setid);
```

Add an object set to MYCOLLECTION to index all the multi-member source physical files library ISVSQLLP:

```
> CALL MYCOLLECTION.ADD_SRCPF_OBJECT_SET('ISVSQLLP','*ALLSRCPF');
```

To add source physical file object set to a collection from IBM® Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management** > **System** > **All Tasks** > **OmniFind** > **Collection List**.
2. Right click the collection and select **Properties**. On the **Object** tab, press pull-down menu, choose **Add Source physical file**, then press **ADD**.

Removing an Object Set for Multiple Members Source Physical File

The stored procedure is in the DB2® schema to remove an object set for multi-member source physical file.

RMV_SRCPF_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

This procedure allows a user to remove an object set of multi-member source physical file (MMPF).

remove an object set for multi-member source physical file:

```
>>-RMV_SRCPF_OBJECT-SET -(---SRCPF_LIB----->  
>----- SRCPF_NAME -----)<
```

The schema qualifier is the name of the text search collection.

Parameter

SRCPF_LIB

This parameter means an absolute library which contains the source physical files.

The library name should be absolute and should not contain any regular expressions.

The data type for this parameter is VARCHAR(10)

SRCPF_NAME

This parameter means an absolute source physical file which can have one or more members, all the members in the source physical file will be indexed. The file name should be absolute and should not contain any regular expressions.

***ALLSRCPF**

If *ALLSRCPF was specified as source physical name, it means will remove the object set for all the source physical files in SRCPF_LIB.

The data type for this parameter is VARCHAR(10)

Result Note

In following case remove will failed due to object set not found:

```
>User inputs incorrect SRCPF_LIB or SRCPF_NAME  
>Specifies SRCPF_LIB/SRCPF_NAME corresponding object set has been deleted previously.
```

User will get error message show that the object set doesn't exist with the specific attribute.

QUERY_OBJECT_SET() returns the object set list and the input parameters. User can specify correct input parameters while invoking this stored procedure to remove the object set.

Examples

Remove an object set in MYCOLLECTION for the multi-member source physical file QCSRC in library ISVSQLLP:

```
> CALL MYCOLLECTION.RMV_SRCPF_OBJECT_SET('ISVSQLLP','QCSRC');
```

Remove an object set in MYCOLLECTION for all the multi-member source physical files library ISVSQLLP:

```
> CALL MYCOLLECTION.RMV_SRCPF_OBJECT_SET('ISVSQLLP','*ALLSRCPF');
```

Removing an Object Set

This stored procedure will remove an object set from a text search collection.

REMOVE_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

```
➤ — REMOVE_OBJECT_SET — ( — setid — ) ➤
```

The schema qualifier is the name of the text search collection.

Parameters

setid

The set id which was obtained when adding object set.

The object set ID can also be obtained using the QUERY_OBJECT_SET stored procedure after the object set has been added.

The data type for this parameter is INTEGER.

Examples

Remove object set #1 from collection MYCOLLECTION.

```
> CALL MYCOLLECTION.remove_object_set(1)
```

To remove an object set from a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Properties**. On the **Object** tab, press **Remove** button.

Update the Collection

This stored procedure updates the collection. When called initially, all objects included in the object sets for the collection are indexed. When this stored procedure is called after a successful initial update has completed, all changed objects are updated in the index. The procedure will not return control to the caller until after the update has completed.

UPDATE

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

➔ — UPDATE ➔

The schema qualifier is the name of the text search collection.

Parameters

None

Authority Requirements on Indexed Objects

Scheduled updates run under the profile that owns the text search collection.

Calls to the UPDATE stored procedures run under the profile invoking the procedure.

It is recommended that both the owner of the index, and the profile administering the index have authority to read the text data from all objects indexed in the collection. Failure to have sufficient authority can cause unpredictable results.

Adopted authority will not necessarily be honored for crawling objects and extracting text from those objects.

Authority problems during the update process may cause the update to fail, or individual documents not to be indexed.

Examples

```
CALL MYCOLLECTION.UPDATE;
```

To update the collection index from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Update**.

Repopulate the Text Search Collection

The REPRIME stored procedure clears the collection, and then performs an initial update. The procedure will not return control to the caller until after the update has completed.

REPRIME

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

➤ — REPRIME ➤

The schema qualifier is the name of the text search collection.

Parameters

None

Reprime Authority Considerations to Indexed Objects

Scheduled updates run under the profile that owns the text search collection.

Calls to the REPRIME stored procedures run under the profile invoking the procedure.

It is recommended that both the owner of the index, and the profile administering the index have authority to read the text data from all objects indexed in the collection. Failure to have sufficient authority can cause unpredictable results.

Adopted authority will not necessarily be honored for crawling objects and extracting text from those objects.

Authority problems during the update process may cause the update to fail, or individual documents not to be indexed.

Examples

```
CALL MYCOLLECTION.REPRIME;
```

To repopulate the text search collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Reprime**.

Search the Collection

This procedure allows a user to search a text search collection for objects that match a search.

SEARCH

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

```
SEARCH ( search_string  
      , search_options  
      , number_of_results  
      )
```

The schema qualifier is the name of the text search collection.

Parameters

search_string

A string parameter that contains the search expression.

Note: This expression must not be all blanks or empty string.

See the Search Argument Syntax of the OmniFind Reference for details.

The data type of this parameter is VARCHAR(32700).

search_options

Identifies a string constant that specifies the search argument options that are in effect for the function.

The data type of this parameter is VARCHAR(32700).

```
search_options  
  , QUERYLANGUAGE = value  
  , RESULTLIMIT = value  
  , SYNONYM = OFF  
  , SYNONYM = ON
```

QUERYLANGUAGE

Specifies the query language. The value can be any of the supported language codes. If the QUERYLANGUAGE option is not specified, the default is the language value of the text search index that is used when this function is invoked. If the language value of the text search index is AUTO, the default value for QUERYLANGUAGE is en_US.

RESULTLIMIT

This provides a clue to the DB2 optimizer for how many rows of the result set are expected to be used. The optimizer may choose a different plan to return fewer rows from the SEARCH procedure. The optimizer may also ignore this option if a performance benefit during the search will not be obtained.

SYNONYM

Specifies whether to use a synonym dictionary that is associated with the text search index. You can add a synonym dictionary to a collection by using the synonym tool. OFF is the default value.

number_of_results

Output Integer value that returns the number of documents for the search result.

The data type of this parameter is VARCHAR(32700).

RESULT SET RETURNED

The search procedure returns a result set with matches for the search expression.

The result set contains following columns:

OBJTYPE	CHAR(10)	CCSID	1208
OBJATTR	CHAR(10)	CCSID	1208
CONTAINING_OBJECT_LIB	CHAR(10)	CCSID	1208
CONTAINING_OBJECT_NAME	CHAR(10)	CCSID	1208
OBJECTINFOR	XML		
MODIFY_TIME	TIMESTAMP		
SCORE	DOUBLE		

OBJTYPE - The type of system object for this result (*STMF, *OUTQ, etc).

OBJATR - The attribute of the system object that matched the search expression (*SPLF, *DATA, etc).

CONTAINING_OBJECT_LIB - The library for the matched system object.

CONTAINING_OBJECT_NAME - The name of the matching system object.

OBJECTINFOR - An XML value that describes the location information of the indexed data that matched the search_string expression. An example spool file location will look like:

```
<Spool_File xmlns="http://www.ibm.com/xmlns/prod/db2textsearch/obj1">
  <job_name>QPADEV000C</job_name>
  <job_user_name>USERA</job_user_name>
  <spool_file_name>DSXSVRALS</spool_file_name>
  <spool_file_number>1</spool_file_number>
  <job_system_name>ZD21BP1</job_system_name>
  <create_date>1081027</create_date>
  <create_time>035554</create_time>
</Spool_File>
```

An example IFS stream file location might look like:

```
<Stream_File xmlns="http://www.ibm.com/xmlns/prod/db2textsearch/obj1">
  <file_path>/home/usera/a.txt</file_path>
</Stream_File>
```

An example physical file member location might look like:

```
<Source_Physical_File_Member xmlns="http://www.ibm.com/xmlns/prod/db2textsearch/obj1">
  <file_library>MYLIB</file_library>
  <file_name>MYPF</file_name>
  <member_name>member1</member_name>
</Source_Physical_File_Member>
```

DB2 provides a number of ways to convert an XML value into other formats so that applications can access the data. One possibility is to create an xsl-stylesheet and use the [XSLTRANSFORM](#) function. Another possibility is to create an annotated schema for the XML values and use the [XDBDECOMPXML](#) procedure to populate relational tables with the values. The SQL Reference in the [infocenter](#) contains details on both of these functions.

MODIFY_TIME - A timestamp indicating the last modification time of the object that is in the collection. This timestamp will never be more recent than the last update process.

SCORE - The result is greater than 0 but less than 1 if the indexed text data contains a match for the search criteria that the search argument specifies. The more frequently a match is found, the larger the result value. If the column does not contain a match, the result is 0.

The result set is ordered by score descending.

Examples

1. call MYCOLLECTION.search('big bad wolf');
2. call MYCOLLECTION.search('big bad wolf', 'QUERYLANGUAGE=en_US');

To search the collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > Search**.
2. Select Search.

Query Object Set Information

This procedure allows a user to see the list of object sets that are contained within the collection.

QUERY_OBJECT_SET

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

➤ — QUERY_OBJECT_SET ➤

The schema qualifier is the name of the text search collection.

Parameters

None

RESULT SET RETURNED

The procedure returns a result set that has one row for each object set:

SETID	INTEGER	
SETSTATE	CHAR(10)	CCSID 1208
LASTREFRESHTIME	TIMESTAMP	
ADDOBJSETSQL	VARCHAR(2000)	CCSID 1208

SETID - A unique identifier assigned to each object set. This identifier may be used on the remove_object_set stored procedure to remove the object set from the collection.

SETSTATE - The state of the object set. This is reserved for future expansion and is always 'ACTIVE'.

LASTREFRESHTIME - The last time the object set has been refreshed to reflect objects actually on the system.

ADDOBJSETSQL - The SQL stored procedure call that was used to add this object set.

Examples

```
Call MYCOLLECTION.QUERY_OBJECT_SET;
```

To query object set information from a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Properties**. Click the **Object** tab.

Retrieve Status of Indexes Objects

This procedure returns the status of all objects in the text search collection.

GET_OBJECT_STATUS

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

➤ — GET_OBJECT_STATUS ➤

The schema qualifier is the name of the text search collection.

Parameters

None

RESULT SET RETURNED

The procedure returns a result set that has one row for each object set:

OBJECT	XML
STATUS_TIMESTAMP	TIMESTAMP
STATUS_CODE	INTEGER
TEXT_STATUS	VARCHAR(100)

OBJECT - The location information for the object in the index, this matches the format returned by search for the OBJECTINFOFOR column.

STATUS_TIMESTAMP - If the object is in the INDEXED state, then this is the modification timestamp of the object in the index. In other words changes made to the object prior to this time are reflected in the text search index. For all other state values, this timestamp is the time the state was recorded in the index. For example: If the object's state is ERROR, this is the time that the error was recorded.

STATUS_CODE - A numeric value representing the state of this object:

0	= The object has been indexed and no changes were detected to the object.
10	= The object is currently in the process of being indexed.
20	= Changes to the object have been detected.
30	= The object was indexed, but a warning occurred during the indexing process.
40	= An error prevented the object from being indexed.

These values allow more complex selection criteria. i.e. retrieve all objects not current in the index could be expressed as STATUS_CODE > 0.

Note: the state of objects that are in the process of being updated, or were in the process of being updated when an update was canceled, could differ from reality. It is possible for an object to report there are changes pending, when in fact the text to be indexed has already been sent to the server. It is also possible to a document's status to be 'INDEXED', but the object may not be search-able until a future commit point in the update process. The STATUS_CODE will reflect reality when the update stored procedure has completed.

The result set is sorted (descending) by status code.

TEXT_STATUS - The Text Version of the status code:

```
0 = INDEXED
10 = INDEXING
20 = CHANGES PENDING
30 = WARNING
40 = ERROR
```

Examples

```
Call MYCOLLECTION.GET_OBJECT_STATUS;
```

Get Objects Not Indexed

This procedure returns the objects which are not indexed in the text search collection.

GET_OBJECTS_NOT_INDEXED

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

```
► — GET_OBJECTS_NOT_INDEXED ◄
```

The schema qualifier is the name of the text search collection.

Parameters

None

RESULT SET RETURNED

The procedure returns a result set that has one row for each object set:

OBJECT	XML
STATUS_TIMESTAMP	TIMESTAMP
STATUS_CODE	INTEGER
TEXT_STATUS	VARCHAR(100)

OBJECT - The location information for the object in the index, this matches the format returned by search for the OBJECTINFOR column.

STATUS_TIMESTAMP - If the object is in the INDEXED state, then this is the modification timestamp of the object in the index. In other words changes made to the object prior to this time are reflected in the text search index. For all other state values, this timestamp is the time the state was recorded in the index. For example: If the object's state is ERROR, this is the time that the error was recorded.

STATUS_CODE - A numeric value representing the state of this object:

```
0 = The object has been indexed and no changes were detected to the object.
10 = The object is currently in the process of being indexed.
20 = Changes to the object have been detected.
30 = The object was indexed, but a warning occurred during the indexing process.
40 = An error prevented the object from being indexed.
```

These values allow more complex selection criteria. i.e. retrieve all objects not current in the index could be expressed as STATUS_CODE > 0.

Note: the state of objects that are in the process of being updated, or were in the process of being updated when an update was canceled, could differ from reality. It is possible for an object to report there are changes pending, when in fact the text to be indexed has already been sent to the server. It is also possible to a document's status to be 'INDEXED', but the object may not be search-able until a future commit point in the update process. The STATUS_CODE will reflect reality when the update stored procedure has completed.

The result set is sorted (descending) by status code.

TEXT_STATUS - The Text Version of the status code:

```
0   = INDEXED
10  = INDEXING
20  = CHANGES PENDING
30  = WARNING and ERROR
40  = WARNING and ERROR
```

Examples

```
Call MYCOLLECTION.GET_OBJECTS_NOT_INDEXED;
```

Retrieve Status of Collection

This procedure returns the status of the text search collection.

STATUS

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

➔ — STATUS ➔

The schema qualifier is the name of the text search collection.

Parameters

None

RESULT SET RETURNED

The procedure returns a result set that has one row for each object set:

SERVERID	INTEGER
SERVERNAME	VARCHAR(128)
SERVERSTATUS	VARCHAR(32)
LASTUPDATETIME	TIMESTAMP
LASTUPDATESTATUS	VARCHAR(30)
UPDATEFREQUENCY	VARCHAR(512)

SERVERID - The server ID for the text search index.

SERVERNAME - The host name or IP address of the text search server.

SERVERSTATUS - Indicates whether the server can be used as a text search server to create new text search indexes. The default value is 0 (zero), which means that the server can be used.

LASTUPDATETIME - The time that the text search index was last updated.

LASTUPDATESTATUS - Indicate last update status for the text search index.

```
NEVER UPDATED - The text search index is never updated before.  
UP TO DATE - The text search index is up to date.  
CHANGES PENDING - There are changes pending. User need update the index to make the text search  
index up to date.  
FAILED - The text search index was updated failed last time.
```

UPDATEFREQUENCY - When to make automatic updates to the text search index.

Examples

```
Call MYCOLLECTION.STATUS;
```

To retrieve status of collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.

Dropping a Text Search Collection

This procedure removes a text search collection from the system.

SYSPROC.SYSTS_DRPCOL and SYSPROC.SYSTS_DROP_COLLECTION

Authorization

The SYSPROC.SYSTS_DRPCOL and SYSPROC.SYSTS_DROP_COLLECTION stored procedures are shipped with public authority *EXECUTE.

No authority is adopted and the procedure runs under the user's profile.

The user must have authority to drop the SQL schema (including all objects within) in order to successfully drop the collection.

Syntax

```
► — SYSTSDRPCOL — ( — collection_name — ) ◄
```

Syntax

```
► — SYSTSDROP_COLLECTION — ( — collection_name — ) ◄
```

The schema qualifier is SYSPROC.

Parameters

collection_name

The name of the collection as supplied on the SYSTSCREATE_COLLECTION stored procedure.

The data type for this parameter is VARCHARE(128).

Examples

```
CALL SYSPROC.SYSTSDRPCOL('MYCOLLECTION')
```

Other examples using text search collection procedures

```
> CALL SYSPROC.SYSTS_CREATE_COLLECTION('MYCOLLECTION', 'FORMAT TEXT');

By default, always called the stored procedures associated with the new collection
> SET SCHEMA MYCOLLECTION
> SET PATH MYCOLLECTION

Add an object set for all spool files owned by user ZOOKEEPER
> CALL.ADD_SPLF_OBJECT_SET('', '', 'ZOOKEEPER');

Add an object set for all spool files created on 06/14/2010
> CALL ADD_SPLF_OBJECT_SET('', '', '', '', '', '', '', '2010-06-14T00:00:00',
'2010-06-15T00:00:00');

Add an object set for all stream files in the IFS directory '/home/zookeeper'
> CALL ADD_IFS_STMF_OBJECT_SET('/home/zookeeper');

Update the collection
> CALL UPDATE;

Search for 'Lions AND tigers AND bears'
> CALL SEARCH('lions AND tigers AND bears');

Grant authority to another user (SEARCHER)
> GRANT EXECUTE ON SPECIFIC PROCEDURE SEARCH1 TO SEARCHER

Drop the collection
> CALL SYSPROC.SYSTS_DROP_COLLECTION('MYCOLLECTION')
```

To drop a text search collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Collection List**.
3. Right click the collection and select **Delete**.

Altering a Text Search Collection

You can call this stored procedure to modify attributes of a collection that was created by SYSPROC.SYSTS_CREATE_COLLECTION. Only attributes explicitly specified on this procedure are changed. All other attributes of the index remain unchanged.

This is useful if you need to change the attributes of the collection, such as the update frequency, after the collection has already been created.

ALTER_COLLECTION

Authorization

This stored procedure is created with public authority *EXCLUDE and is owned by the creator of the text search collection.

The procedure will adopt the authority of the text search collection owner's profile. Authority can be granted to other users to allow them to execute the procedure.

Syntax

```
>>-ALTER_COLLECTION--(--options-----><
```

The schema qualifier is the name of the text search collection.

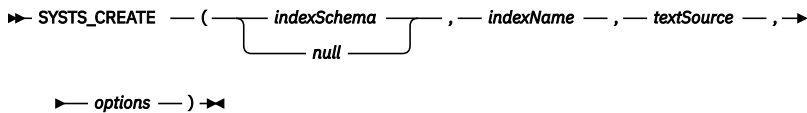
Parameters

options

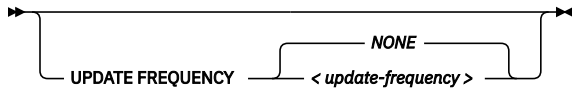
A character string that specifies the various options that are available for this stored procedure.

The data type for this parameter is VARCHAR(32000).

options



update-characteristics



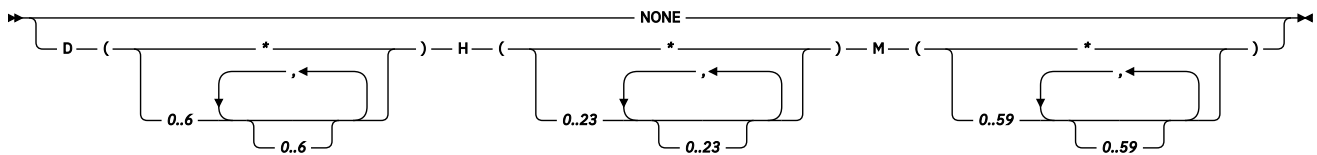
update-characteristics

Specifies the frequency of automatic updates to the text search collection. The update process for a text search collection involves both indexing the text data, and crawling system objects to detect new or changed data.

UPDATE FREQUENCY update-frequency

Specifies when to make automatic updates to the text search collection. The default value is NONE.

update-frequency (Format 1)



NONE

If NONE is specified, then no further index updates are made. The update must be started manually.

D

Specifies the day or days of the week when the index is updated. An asterisk (*) specifies all days. 0 specifies Sunday.

H

Specifies the hour or hours when the index is updated. An asterisk (*) specifies all hours.

M

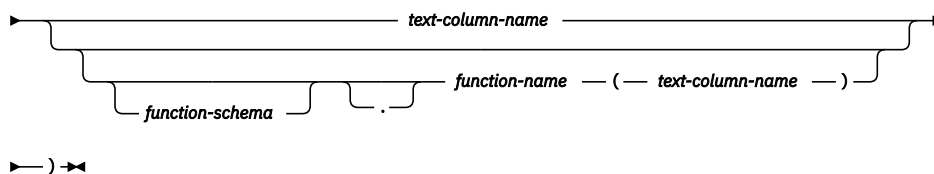
Specifies the minute or minutes when the index is updated. An asterisk (*) cannot be specified. The minimum update frequency is 5 minutes.

Example: This example specifies that the index update is to run every 30 minutes.

```
UPDATE FREQUENCY D(*) H(*) M(0,30)
```

update-frequency (Format 2, chronological)

► *tableSchema* . *tableName* — (→



The format of the *update-frequency (chronological)* option is a list of the five values separated by a blank space. The five values represent the minutes, hours, days of the month, months of the year, and days of the week beginning with Sunday.

If you specify an interval of values or an asterisk (*), you can specify a step value by using a forward slash (/) at the end of the defined interval.

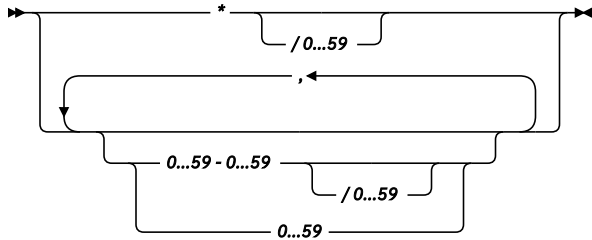
Example: This example specifies that the index update is to run every quarter hour (0, 15, 30, 45) on the even hours between 8 a.m. and 6:45 p.m. (8-18/2 is equivalent to 8, 10, 12, 14, 16, 18), from Monday to Friday every month of the year (* * 1-5).

```
0,15,30,45 8-18/2 * * 1-5
```

minute

Specifies the minutes of the hour when the text search index is to be updated. You can specify an asterisk (*) for an interval of every 5 minutes, or you can specify an integer 0 - 59. You cannot repeat values. The minimum update frequency is 5 minutes. A value of 1,4,8 is not valid.

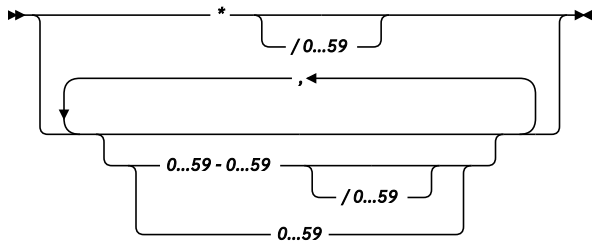
update-frequency (minute)



hour

Specifies the hours of the day when the text search index is to be updated. You can specify an asterisk (*) for every hour, or you can specify an integer 0 - 23. You cannot repeat values.

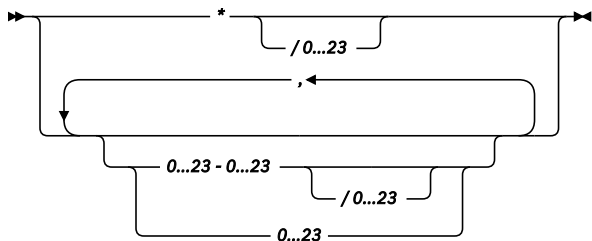
update-frequency (hour)



dayOfMonth

Specifies the days of the month when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 1 - 31. You cannot repeat values.

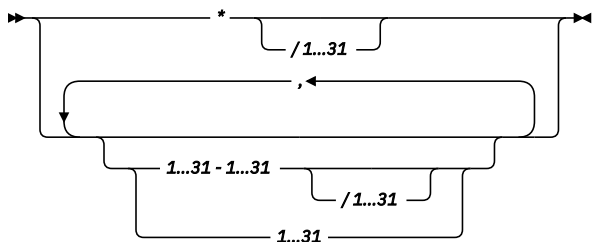
update-frequency (dayOfMonth)



monthOfYear

Specifies the months of the year when the text search index is to be updated. You can specify an asterisk (*) for every month, or you can specify an integer 1 - 12. You cannot repeat values.

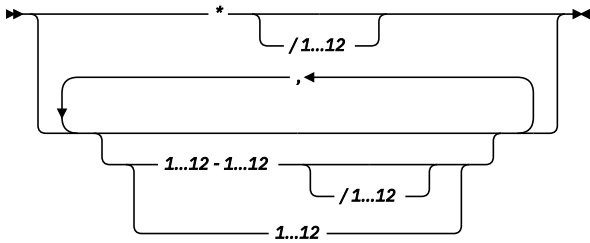
update-frequency (monthOfYear)



dayOfWeek

Specifies the days of the week when the text search index is to be updated. You can specify an asterisk (*) for every day, or you can specify an integer 0 - 7. Both 0 and 7 are valid values for Sunday. You cannot repeat values.

update-frequency (dayOfWeek)



Examples

1. CALL MYCOLLECTION.ALTER_COLLECTION('UPDATE FREQUENCY H(0) M(0) D(*)');

The update frequency value of collection MYCOLLECTION is changed. The text search collection will have an update frequency of every day, at 00:00.

2. CALL MYCOLLECTION.ALTER_COLLECTION('UPDATE FREQUENCY NONE');

This text search collection is changed to not scheduled.

To alter a collection from IBM Navigator for i, follow these steps.

1. From IBM Navigator for i, expand **IBM i Management > System > All Tasks**.
2. On the right panel, select **System > OmniFind > Alter Collection**.

Independent ASP Considerations for Text Search Collections

IASP considerations for a text search collection.

Text Search Collections may be created in an Independent Auxiliary Storage Pool (ASP) environment with the following restrictions:

1. If an Independent ASP Group is associated with the current thread, then the collection must reside on an ASP in the independent ASP Group. It is not possible to administer a text search collection in the system ASP or a basic user ASP while the thread is set to an Independent ASP group. It is possible to search a collection that exists in the system ASP or basic user ASP, however because that index cannot include data on the Independent ASP group, only objects that are accessible when the ASP Group is *NONE will be included.
2. A text search collection can index any objects visible from the ASP Group Namespace of the collection.
3. If an object set includes objects that exist within the namespace of the ASP group, but not on the group itself, significant indexing time can occur if the Independent ASP Group is moved to another system. For example: If a collection is created on Independent ASP 33 to index all spool files on the system, and the ASP group is switched to a different machine, then all spool files from the old machine not in the ASP group will be removed from the index, and all spool files on the new machine not in the index will be added.

Backup and Restore Considerations for Text Search Collections

Saving and restoring a text search collection.

A text search collection may be backed up and restored by saving and restoring the library of the schema created for the collection. During the restore, an update will be initiated asynchronously. The update will crawl the objects on the system, and repopulate the index.

It is necessary to be able to restore the text search index contained within the collection in order for the collection to be usable. These considerations for restoring a text search index must be considered:

- All required products must be installed.
- The text search server should be available.

Messages and codes

You can see the messages and SQL return codes for OmniFind Text Search for DB2 for i. The messages are listed in numeric sequence.

OmniFind messages

You can see the OmniFind messages for OmniFind Text Search for DB2 for i. The messages are listed in numeric sequence.

Messages are added to the OmniFind message file (QDBTSLIB/QOMFMSGF) for the following errors.

Number	Type	Message
OMF0011	Information	Text search index restored with different configuration options.
OMF0012	Warning	The FORMAT type for the index being created is not XML. XML searches are not supported.
OMF0334	Error	The object "{0}" "{1}" you specified is not supported. A Text Search index can only be built over an SQL table, an SQL alias, or a single member of a physical file.
OMF0358	Error	The current user does not have enough authority to perform the requested operation.
OMF0359	Error	Restore failed. The text server for the text index is not available.

SQLCODE **Column &3 in &1 in &2 cannot be**
-0196 **dropped.**

Explanation:

An attempt was made to drop column &3. The column cannot be dropped because a view, a constraint, a trigger, or an index is dependent on the column and RESTRICT was specified, or the column is part of the partition key.

User response

Specify CASCADE on the ALTER TABLE statement to drop the column and the views, constraints, triggers, and indexes that are dependent on it. If the column is part of the partition key, specify DROP PARTITIONING on the ALTER TABLE statement to remove the partitioning for the table. Try the request again.

SQLSTATE: 42817

SQLCODE **Cannot perform operation under**
-5003 **commitment control.**

Explanation

The following operations cannot be performed under commitment control with COMMIT(*CHG), COMMIT(*CS), or COMMIT(*ALL) specified:

- DROP SCHEMA statement.
- GRANT or REVOKE statement to an object that has an authority holder.
- CREATE statement in SQL naming mode of an object that has an authority holder.
- DROP of a text search index.

These operations cannot be committed or rolled back.

User response

Specify COMMIT(*NONE), and try the statement again.

SQLSTATE: 42922

SQLCODE **Error occurred during text search**
-20423 **processing.**

Explanation

An error occurred during the text search processing of a CONTAINS or SCORE function. The error happened

on server *server* using text search index *index-name* for reason code *reason-code*. Text describing the problem is:*text*.

server: The host name or IP address and port of the text search server where the error was encountered.

index-name: The name of the index used in the text search processing.

Note: Include the schema and a period with the index name in a single token.

reason-code: The reason code returned from the OmniFind Text Search Server for DB2 for i.

text: The text returned from the OmniFind Text Search Server for DB2 for i.

System action:

The statement cannot be processed.

User response

Contact your system administrator to check that the OmniFind Text Search Server for DB2 for i is successfully installed.

SQLSTATE: 38H10

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

SQLCODE -20424	Text search support is not available for reason <i>reason-code</i>.
---------------------------------	--

Explanation

A problem with one of the text search administrative tables was detected. The reason code is *reason-code*.

1

One of the text search administration tables was not found (QSYS2.SYSTEXTINDEXES, QSYS2.SYSTEXTCOLUMNS, or QSYS2.SYSTEXTSERVERS).

3

The Text Search support is not started.

4

The STATUS column in QSYS2.SYSTEXTSERVERS table has a value of 1, indicating that the support for the text search is stopped.

7

No OmniFind Text Search Server for DB2 for i have been defined.

System action:

The statement cannot be processed.

User response

Contact your system administrator to make sure that support for text searching is successfully set up on your system.

SQLSTATE: 38H11

Related reference

QSYS2.SYSTEXTSERVERS administration table

You can see where the text search servers are installed using the QSYS2.SYSTEXTSERVERS administration table.

QSYS2.SYSTEXTINDEXES administration table

You can see information about each text search index in the QSYS2.SYSTEXTINDEXES administration table. Each text search index has a name, schema name, and an associated collection name on the text search server.

QSYS2.SYSTEXTCOLUMNS administration table

You can see information about the text columns for a text search index in the QSYS2.SYSTEXTCOLUMNS administration table. Each text search index has an index ID, text column names, and the schema name of the base table.

SQLCODE -20425	Text search not allowed for column <i>column-name</i>.
---------------------------------	---

Explanation:

A CONTAINS or SCORE text search function specified column *column-name* in table *table-name* in table-schema. A text index does not exist for this column so text search processing cannot be performed.

System action:

The statement cannot be processed.

User response

Verify that the column and table are registered to the OmniFind Text Search Server for DB2 for i.

SQLSTATE: 38H12

Related reference

CONTAINS

You can use the CONTAINS function to search a text search index using criteria you specify in a search argument. The function returns a result indicating whether a match was found.

SCORE

You can use the SCORE function to search a text search index using criteria that you specify in a search argument. The function returns a relevance score that measures how well a document matches the query.

SQLCODE -20426	Conflicting text search administration procedure is already running.
---------------------------------	---

Explanation:

A conflicting text search administrative procedure such as update is already running on this index.

System action:

The statement cannot be processed.

User response

Invoke the administration stored procedure again after the currently running stored procedure completes.

SQLSTATE: 38H13

SQLCODE -20427	Error occurred during text search administrative procedure.
---------------------------------	--

Explanation:

An error occurred during a text search administrative procedure. The reason code is *reason-code*. The text returned is: *text*. The error text describes the problem.

System action:

The CALL statement fails with this SQLCODE.

User response

Fix the problem that is indicated by *error* and invoke the administrative stored procedure again.

SQLSTATE: 38H14

CPF32fa	Operation not allowed on text search index &2 in &1.
----------------	---

Explanation

An operation was attempted that is not supported for a text search index. Text search indexes do not allow some operations that are allowed for traditional DB2 indexes and views.

If this was an attempt to delete the index, the operation may have failed because commitment control was active.

User response

Perform text search administrative operations using the administrative SQL stored procedures that are included with OmniFind Text Search Server for DB2 for i.

For more information on text search indexes, and what restrictions apply to them, refer to the

documentation in the Information Center: <http://www.ibm.com/systems/i/infocenter/>

CPF32fb	Operation on text search index &2 in &1 could not be completed.
----------------	--

Explanation

An operation was attempted against text search index &2 in &1. The requested operation is not currently valid for reason code &3 reason codes and their meanings are:

1. A required product is not installed
2. The requested text search server &4 is not available or is not defined.
3. A restore of the index was attempted but the index already exists. The existing index could not be modified to match the saved index.
4. A dependent object &5 in &6 type &7 did not exist.
5. An object &5 in &6 type &7 was not available.
6. The auxiliary storage pool (ASP) for the current thread does not match the ASP of the text search index.
7. A text search index already exists for column &8, table &5 in library &6.

User response

Correct the problem and retry the operation.

For more information on text search indexes, refer to the documentation in the Information Center: http://www.ibm.com/systems/i/infocenter

CPI321E	File &1 in library &2 deferred.
----------------	--

Explanation

File &1 in library &2 was deferred during this restore request with specified Defer ID of &4. The file was deferred because of reason code &3. The reason codes are:

1. Based-on file &5 in library &6 did not exist when &1 was being created for the restore.
2. File &1 failed to create for some other reason than a missing based-on file. See previous message(s) to determine why the create of the file failed.
3. One or more of the members for file &1 failed to create. See previous message(s) to determine why the create of the member(s) failed.
4. The file represents a Text Search Index, and the required licensed program objects do not exist on the system.
5. The File represents a Text Search Index, and the index could not be recreated.

User response

For reason code 1: Either restore the missing based-on file, or use the Restore Deferred Objects (RSTDFROBJ) command specifying the same Defer ID (DFRID parameter) &4 on either of the commands used.

For reason codes 2 and 3: Correct the reasons for the failed create, and then use the Restore Deferred Objects (RSTDFROBJ) command specifying the same Defer ID (DFRID parameter) &4 on the command.

For reason code 5: See the previous messages in the job log, correct any errors, and then use the Restore Deferred Objects (RSTDFROBJ) command, specifying the same Defer ID (DFRID parameter) &4 on the command.

Related information

[Restore Deferred Objects \(RSTDFROBJ\)](#)

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Common Information Model (CIM) publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of the IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be

trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Index

A

- add an object set for a stream file
 - [ADD_IFS_STMF_OBJECT_SET 124](#)
- add an object set for spool file data
 - [ADD_SPLF_OBJECT_SET 117](#)
- [ADD_SPLF_OBJECT_SET 117, 124](#)
- administration tables
 - [QSYS2.SYSTEXTCOLUMNS 108, 111](#)
 - [QSYS2.SYSTEXTCONFIGURATION 110](#)
 - [QSYS2.SYSTEXTINDEXES 106](#)
 - [QSYS2.SYSTEXTSERVERHISTORY 110](#)
 - [QSYS2.SYSTEXTSERVERS 109](#)
 - [SYSIBMTS.SYSTEXTDEFAULTS 105](#)
- Administration Tool [84](#)
- advanced search operators
 - [CONTAINS function 43](#)
 - [SCORE function 43](#)
- asynchronous indexing [5](#)

C

- client and server communication [2](#)
- [COLLECTION_NAME.UPDATE 131](#)
- [collection.status 138](#)
- command line tools
 - Administration Tool [84](#)
 - Configuration Tool [70](#)
 - Health Checker [86](#)
 - ServerInstance Tool [85](#)
 - Synonym Tool [81](#)
- Configuration Tool [70](#)
- configure [11](#)
- [CONTAINS 43](#)
- [CONTAINS function](#)
 - [example 48](#)

D

- data types
 - supported [6](#)
- dictionary pack [7](#)
- dictionary-based segmentation [8](#)
- document formats
 - supported [5](#)
- document size [13](#)
- document truncation [13](#)
- dropping a text search collection
 - [SYSTS_DROP_COLLECTION 139](#)

E

- EBNF grammar [63](#)
- extensions header
 - text search collections [112](#)

F

- functions
 - [CONTAINS 36](#)
 - [SCORE 39](#)

G

- Get Objects Not Indexed [137](#)
- [Get_object_status 136](#)
- [GET_OBJECTS_NOT_INDEXED 137](#)

H

- hardware
 - requirements [3](#)
- Health Checker [86](#)
- high availability [88](#)

I

- IASP for Text Search Collections
 - Text Search Collections [143](#)
- IBM OmniFind Text Search Server
 - starting [64](#)
 - stopping [65](#)
- IBM text search server [3](#)
- install [11](#)
- installation [11](#)

K

- key concepts
 - text search functions [4](#)

L

- language codes [7](#)
- languages
 - supported [7](#)
- linguistic processing
 - Chinese
 - linguistic processing [8](#)
 - Japanese
 - linguistic processing [8](#)
 - Korean
 - linguistic processing [8](#)
- log files [69](#)

M

- messages [144](#)

N

n-gram segmentation [7](#), [8](#)

O

OmniFind [2](#), [11](#)

OmniFind Text Search [11](#)

OmniFind text search server

OmniFind [2](#)

text index [2](#)

text search [2](#)

OmniFind Text Search Server [11](#)

operating system

requirements [3](#)

P

performance [89](#)

populating [11](#)

problem determination [69](#)

product overview [2](#)

Q

QDBTS_LISTINXSTS

UDTF [83](#)

QDBTS_LISTINXSTS UDTF [83](#)

QSYS2.SYSTEXTCOLUMNS [108](#), [111](#)

QSYS2.SYSTEXTCONFIGURATION [110](#)

QSYS2.SYSTEXTINDEXES [106](#)

QSYS2.SYSTEXTSERVERHISTORY [110](#)

QSYS2.SYSTEXTSERVERS [11](#), [109](#)

query examples

CONTAINS function [43](#)

SCORE function [43](#)

query information about object sets [135](#)

QUERY_OBJECT_SET [135](#)

R

relevance score [39](#)

removing an object set

REMOVE_OBJECT_SET [130](#)

repopulate the text search collection [132](#)

reprime collection [132](#)

restore [66](#), [67](#)

Restore for Text Search Collections

Restore Text Search Collections [143](#)

retrieve status of collection [138](#)

retrieve status of indexed objects [136](#)

S

save [66](#), [67](#)

save and restore [66](#), [67](#)

SCORE [43](#)

SCORE function

example [48](#)

search [89](#), [92](#)

search argument [36](#), [39](#)

search argument syntax
excluding terms [41](#)

search argument syntax (*continued*)

including terms [41](#)

phrase search [41](#)

simple search [41](#)

wildcard character [41](#)

search collection

SEARCH [132](#)

SYSPROC.SYSTS_CRTCOL [112](#)

search syntax [36](#)

server log

saving [69](#)

viewing [69](#)

ServerInstance Tool [85](#)

software

requirements [3](#)

SQL return codes [144](#)

stored procedure

SYSPROC.SYSTS_REMOVE [74](#)

SYSPROC.SYSTS_REPRIMEINDEX [76](#), [77](#)

SYSPROC.SYSTS_VALIDITYCHECK [78](#)

stored procedures

SYSPROC.SYSTS_ALTER [26](#)

SYSPROC.SYSTS_CREATE [17](#)

SYSPROC.SYSTS_DROP [31](#)

SYSPROC.SYSTS_START [14](#)

SYSPROC.SYSTS_STOP [15](#)

SYSPROC.SYSTS_UPDATE [32](#)

synonym dictionary

adding [81](#)

removing [82](#)

synonym support [6](#)

Synonym Tool [81](#)

SYSIBMTS.SYSTXTDEFAULTS [105](#)

SYSPROC.SYSTS_CREATE [17](#)

SYSPROC.SYSTS_CREATE_COLLECTION [112](#)

SYSPROC.SYSTS_CRTCOL [112](#)

SYSPROC.SYSTS_DROP [26](#), [31](#)

SYSPROC.SYSTS_START [14](#)

SYSPROC.SYSTS_STOP [15](#)

SYSPROC.SYSTS_UPDATE [32](#)

system requirements [3](#)

SYSTEXTSERVERS [11](#)

T

text index [2](#), [66](#), [67](#)

text score [6](#)

text search

starting [12](#)

text search collections

extensions header [112](#)

Text Search for DB2 for i [11](#)

text search functions

key concepts [4](#)

text search index

creating [12](#)

creation [4](#), [112](#)

searching [13](#)

updates [4](#), [112](#)

updating [13](#)

text search servers [11](#)

transaction [92](#)

triggers [5](#)

U

UPDATEAUTOCOMMIT [89, 92](#)
updating the collection [131](#)
user-defined functions [36](#)

X

XML data
 indexing [5](#)
XML search
 features [49](#)
 query grammar [63](#)
XPath language [63](#)



Product Number: 5733-OMF