Using InterSystems SQL Search

Version 2024.2
2024-09-05

InterSystems Corporation   1 Memorial Drive   Cambridge MA 02142   www.intersystems.com

Using InterSystems SQL Search
PDF generated on 2024-09-05
InterSystems IRIS® Version 2024.2

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)

Tel:	+1-617-621-0700
Tel:	+44 (0) 844 854 2917
Email:	support@InterSystems.com

Table of Contents

1
SQL Search

The InterSystems IRIS® SQL Search tool integrates with the InterSystems IRIS Natural Language Processor (NLP) to perform context-aware text search operations on unstructured data, with capabilities such as fuzzy search, regular expressions, stemming, and decompounding. Searching is performed using a standard SQL query with a WHERE clause containing special InterSystems SQL Search syntax.

1.1 Try it

Searching Unstructured Text Data

1.2 Read all about it

InterSystems IRIS Basics: SQL Search

Using InterSystems SQL Search

InterSystems SQL Overview

1.3 Explore more

Text Analytics (Natural Language Processing)

Using InterSystems SQL

2

InterSystems IRIS Basics: SQL Search

This topic introduces you to InterSystems IRIS® support for SQL text search, which provides semantic context searching of unstructured text data in a variety of languages. It covers the following topics:

To use SQL Search in a hands-on online exercise, see Searching Unstructured Text Data. For the full documentation on SQL Search, see the Using SQL Search.

SQL Search presupposes that you know what you are looking for. As such, it is distinct from Natural Language Processing (NLP) analyses of unstructured text data, which often require no prior knowledge of the text contents.

2.1 Why SQL Search Is Important

Important: SQL Search features that require Semantic and Analytic indexes (entity- and language-aware search) use InterSystems IRIS Natural Language Processing (NLP), which InterSystems has deprecated. As a result, these features may be removed from future versions of InterSystems products. The following documentation is provided as reference for existing users only. Existing users who would like assistance identifying an alternative solution should contact the WRC.

The ability to rapidly search unstructured text data is fundamental to accessing the content of the huge volume of text commonly stored by many companies and institutions. Any search facility for such data must have the following functionality:

- Fast search: InterSystems IRIS SQL Search can rapidly search large quantities of data because it is searching a generated optimized index to the data, rather than sequentially searching the data itself.
- Word-aware search: SQL Search is not a string search, it is a search based on semantic structures in the text. The most basic semantic structure for SQL Search is the word. This reduces the number of false positives that result when a string search finds a string embedded in another word, or when a string bridges two words.
- Entity-aware search: SQL Search takes into account multiple words that are grouped by semantic relationship to form entities. It can thus search for multiple words in a specified order (a positional phrase), words appearing within a specific proximity to each other (regardless of sequence), and words found at the beginning or at the end of an entity. This enables you to narrow a search to a word (or phrase) found in a specified context of other words.
- Language-aware search: identifying semantic relationships between words is language-specific. SQL Search contains semantic rules (language models) for ten natural languages. It also provides support for other languages. It does not require the creation or association of dictionaries or ontologies.
- Pattern matching: SQL Search provides both wildcard matching and regular expression (RegEx) matching to match character patterns.

- Fuzzy matching: SQL Search provides fuzzy search for near-matches that take into account a calculated degree of variation from the search string. This enables matching of spelling errors, among other things.
- Derived matching: SQL Search can use decompounding to match root words and component words. SQL Search can use synonym tables to match synonym words and phrases.

2.2 How InterSystems IRIS Implements SQL Search

Important: Semantic and Analytic SQL Search indexes use InterSystems IRIS Natural Language Processing (NLP), which InterSystems has deprecated. As a result, these features may be removed from future versions of InterSystems products. The following documentation is provided as reference for existing users only. Existing users who would like assistance identifying an alternative solution should contact the WRC.

SQL Search can search text data found in a column in an SQL table. In order to do this, you must create an SQL Search index for the column containing the text data. InterSystems implements a table column as a property in a persistent class.

There are three levels of index available, each supporting additional features as well as all of the features of the lower levels: Basic, Semantic, and Analytic:

- *Basic* supports word search and positional phrase search, including the use of wildcards, ranges between words in a phrase, regular expression (RegEx) matching, and co-occurrence search.
- *Semantic* supports all of the Basic functionality, and also supports InterSystems IRIS Natural Language Processing (NLP) entities. It can search for entities, and words or phrases that begin an entity or end an entity. It recognizes NLP attributes, such as negation.
- *Analytic* supports all of the Semantic functionality, and also supports NLP paths. It can also search based on NLP dominance and proximity scores.

Populating the index. Like all SQL indexes, you can either build the index directly after the table has been populated with data, or have SQL automatically build the index entries as you insert records into an empty table. In either case, SQL automatically updates this index as part of subsequent insert, update, or delete operations.

You perform an SQL Search you write a SELECT query in which the WITH clause contains `%ID %FIND search_index()` syntax. The **search_index()** function parameters include the name of the SQL Search index and a search string. This search string can include wildcard, positional phrase, and entity syntax characters. The search string can also include AND, OR, and NOT logical operators.

2.3 Learn More About SQL Search

InterSystems has other resources to help you learn more about SQL Search, including:

- Searching Unstructured Text Data
- Using InterSystems SQL Search

3

Using SQL Search

Important: Semantic and Analytic SQL Search indexes use InterSystems IRIS Natural Language Processing (NLP), which InterSystems has deprecated. As a result, these features may be removed from future versions of InterSystems products. The following documentation is provided as reference for existing users only. Existing users who would like assistance identifying an alternative solution should contact the WRC.

This topic describes the InterSystems SQL Search facility, a tool for performing context-aware text search operations. To use InterSystems SQL Search you must define an SQL Search index for each column containing text that you wish to search. You can then search the text records using a standard SQL query with a WHERE clause containing InterSystems SQL Search syntax. The query will return all records that contain the specified search item or items. You can also highlight the matching text in the returned records.

3.1 Before You Begin

To use Semantic and Analytic SQL Search indexes as described in the examples in this page, you need to have an InterSystems IRIS® instance which is running and is licensed to use the InterSystems IRIS Natural Language Processor (NLP). (You can view the licence key from the Management Portal; select **System Administration** then **Licensing**.)

Examples in this documentation use data from the Aviation.Event SQL table. If you wish to use this sample data, it is available at https://github.com/intersystems/Samples-Aviation. (You do not need to know anything about GitHub or have a GitHub account.) Locate the contents of the README.md file, which appears below the filenames and directories included in the GitHub repository. Scroll to the “Setup instructions” section at the bottom of the README.md file and complete the steps.

3.2 Indexing Sources for SQL Search

Important: Semantic and Analytic SQL Search indexes use InterSystems IRIS Natural Language Processing (NLP), which InterSystems has deprecated. As a result, this feature may be removed from future versions of InterSystems products. The following documentation is provided as reference for existing users only. Existing users who would like assistance identifying an alternative solution should contact the WRC.

You can use SQL Search to search text in %String data type or %Stream.GlobalCharacter (character stream) data type.

To perform a SQL search, the column to be searched must have a defined SQL Search bitmap index. There are four levels of SQL Search indexes. These levels are defined in nested subclasses. Each index level provides all of the features of the previous level, plus additional SQL Search features specific to that level. You can create any of the following SQL Search index types:

- Minimal index (%iFind.Index.Minimal): supports SQL word and word phrase search with wildcards, fuzzy search, and regular expressions. It does not support co-occurrence or positional phrase searches or the highlighting of search results.
- Basic index (%iFind.Index.Basic): supports SQL word and word phrase search with wildcards. It supports co-occurrence and positional phrase searches and highlighting of search results.
- Semantic index (%iFind.Index.Semantic): supports SQL search of NLP entities and optionally supports the negation attribute.
- Analytic index (%iFind.Index.Analytic): supports the all of the NLP features of Semantic index, as well as path, proximity, and dominance information.

Each index level supports all of the parameters of the previous level, and adds one or more additional parameters. Unspecified parameters take default values.

The following Class Definition example creates a table with a Semantic index on the Narrative property (column). The indexed property can be of data type %String or %Stream.GlobalCharacter:

Class Definition

```
 Class Aviation.TestSQLSrch Extends %Persistent [
    DdlAllowed,Owner={UnknownUser},SqlRowIdPrivate,
    SqlTableName=TestSQLSrch ]
{
Property UniqueNum As %Integer;
Property CrashDate As %TimeStamp [ SqlColumnNumber=2 ];
Property Narrative As %String(MAXLEN=100000) [ SqlColumnNumber=3 ];
Index NarrSemanticIdx On (Narrative) As %iFind.Index.Semantic(INDEXOPTION=0,
   LANGUAGE="en",LOWER=1);
Index UniqueNumIdx On UniqueNum [ Type=index,Unique ];
}
```

An SQL Search index of any type includes support for the following parameters:

- IGNOREPUNCTUATION specifies whether or not to ignore punctuation characters. For %iFind.Index.Minimal the default is 1: punctuation is ignored. For all other SQL Search index types, the default is 0: punctuation affects search results; leading and trailing punctuation in the text must match the same punctuation in the search string.
- INDEXOPTION specifies whether or not to index to allow for stemming or decompounding. Because indexing to support these operations significantly increases the size of the index, it is recommended that you specify INDEXOPTION=0 unless stemming or decompounding will very likely be used. The default is 0.
- LANGUAGE specifies the language to use when indexing records. For example, `"en"` specifies English. Use `"*"` to enable automatic language identification. The default is `"en"`.
- LOWER specifies whether query search is case-sensitive. By default, InterSystems SQL Search indexing is not case-sensitive; SQL Search normalizes the text to lowercase before indexing it. The LOWER parameter determines whether or not to perform this lowercase normalization (LOWER=1, the default, normalizes to lowercase). Because language conventions commonly capitalize words at the beginning of a sentence or when used in a title, normalizing to lowercase is preferable in most applications. If you specify LOWER=0, the query *search_items* string is case-sensitive. For example, if you specify LOWER=0, the query *search_items* string ‘turkey’ will only match turkey and not Turkey. If you specify LOWER=1, the query *search_items* string ‘turkey’ will match both turkey and Turkey.
- USERDICTIONARY allows you to specify the name of a user-defined UserDictionary that is applied to the texts before indexing. This parameter is optional and is for advanced use only.

For a full list of supported parameters refer to %iFind.Index.Basic in the *InterSystems Class Reference*.

A Semantic index (%iFind.Index.Semantic) also supports the following optional parameter:

- IFINDATTRIBUTES allows you to specify whether or not to identify negation in the text and store the negation attribute. IFINDATTRIBUTES=1 identifies and indexes negation. The default is IFINDATTRIBUTES=0.

3.2.1 Populating a Table

Like any SQL index, a defined SQL Search index (by default) is built when you populate a new table, and maintained when you subsequently insert, update, or delete data. You can defer building of an index when populating a table using %NOINDEX, and then use the **%Build()** method to build the index. You can add an index to a table that is already populated with data and then build that index. Refer to Defining and Building Indexes for further details.

The following example populates the Aviation.TestSQLSrch table from the Aviation.Events table. Any defined SQL Search indexes will automatically be built. This example inserts a large amount of text, so running it may take a minute or so:

SQL

```
INSERT OR UPDATE INTO Aviation.TestSQLSrch (UniqueNum,CrashDate,Narrative)
    SELECT %ID,EventDate,NarrativeFull FROM Aviation.Event
```

This example uses **INSERT OR UPDATE** with a field defined with a unique key to prevent repeated execution from creating duplicate records.

3.3 Performing SQL Search

You use SQL Search syntax in an SQL query WHERE clause to perform a text search for one or more text items. These text items may be words or sequences of words (Basic index) or NLP semantic entities (Semantic index). Multiple text items are an implicit AND search; all of the specified items must appear in the text, in any order. The syntax for SQL Search is as follows:

```
WHERE %ID %FIND search_index(indexname,'search_items',search_option,'language','synonym_tables')
```

- *indexname* is the name of a defined SQL Search index for a specific column.
- *search_items* is the list of text items (either words or NLP entities) to search for, enclosed with quotes. Text items are separated by spaces. An item consists of an alphanumeric string and optional wildcard syntax characters. By default, text items are not case-sensitive (see the LOWER parameter). Available search_items syntax is described below.
- *search_option* is the index option integer that specifies the type of search to perform. Available values include 0 (syntax search), 1 (syntax search with stemming), 2 (syntax search with decompounding and stemming), 3 (syntax search with fuzzy search), and 4 (syntax search with regular expressions). If *search_option*=4, *search_items* is assumed to contain a single Regular Expression string. For further details, see Regular Expressions. Note that for performance reasons, SQL Search does not support some of the more esoteric regular expression syntactic forms; these syntactic forms are supported by the $LOCATE ObjectScript function.
- *language* is the NLP-supported language model to apply, specified as a two-character string. For example, `'en'` specifies English. If you specify `'*'`, the query performs automatic language identification.
- *synonym_tables* is a string containing the name of a synonym table, or a comma-separated list of synonym tables.

When performing an Basic index search, SQL Search identifies words by the presence of one or more space characters. Sentence punctuation (a period, comma, semicolon, or colon followed by a space) is ignored. SQL Search treats all other punctuation as literals. For example, SQL Search treats “touch-and-go” as a single word. Punctuation such as hyphens or a decimal point in a number are treated as literals. Quote characters and apostrophes must be specified. You specify a single quote character by doubling it.

Be default, text is normalized to lowercase letters (not case sensitive). Other language-specific normalizations, such as normalizing Japanese characters that differ only in character width, are also performed.

You can perform any Basic index search (word, co-occurrence, or positional phrase) with a Semantic index. Attempting to perform a Semantic index search with a Basic index results in an SQLCODE -149 error.

3.3.1 SQL search_items Syntax

Basic index *search_items* can contain the following syntax:

Word Search:

Argument	Description
word1 word2 word3	Specifies that these exact words must appear (in any order) somewhere in the text. (Logical AND). You can specify a single word, or any number of words separated by spaces.
word1 OR word2 NOT word3 word1 OR (word2 AND word3)	*search_items* can contain AND, OR, and NOT logical operators. AND is the same as separating words with spaces (implicit AND). NOT is logically equivalent to AND NOT. *search_items* can also use parentheses to group logical operators. Explicit AND is needed when specifying multiple words in grouping parentheses: (word2 AND word3). If the explicit AND was omitted, (word2 word3) would be interpreted as a positional phrase. You can use the \ escape character to specify AND, OR, NOT as literals rather than logical operators: \and
?word word? w?rd w??d	A question mark wildcard specifies exactly one non-space character of any type. One or more ? wildcards can be used as a prefix, suffix, or within a word. You can combine ? and * wildcards. You can use \ escape character to specify ? as a literal: \?
word word *word* w*d	An asterisk wildcard specifies 0 or more non-space characters of any type. An asterisk can be used as a prefix, suffix, or within a word. You can use \ escape character to specify * as a literal: *

Co-occurrence Word Search:

Argument	Description
[word1,word2,...,range]	Co-occurrence search. Specifies that these exact words must appear (in any order) within the proximity window specified by *range*. You can specify any number of words or multi-word phrases. A multi-word phrase is specified as words separated by spaces with no delimiting punctuation. Words (or positional phrases) are separated by commas, the last comma-separated element is an optional numeric *range*. Words can specify asterisk wildcards. A *range* can be specified as min–max or simply as max with a default min of 1. For example, 1–5 or 5. *range* is optional; if omitted, it defaults to 1–20. A range count is inclusive of all of the specified words. Co-occurrence search cannot be used with *search_option*=4 (Regular Expressions).

Positional Phrase Search:

Note: You can use double quotes `"word1 word2 word3"` or parentheses `(word1 word2 word3)` to delimit a positional phrase. Because parentheses are also used to group logical operators, the use of double quotes is preferred.

Argument	Description
"word1 word2 word3"	These exact words must appear sequentially in the specified order. Words are separated by spaces. Note that no semantic analysis is performed; for example, the words in a “phrase” may be the final word of a sentence and the beginning words of the next sentence. Asterisk wildcards can be applied to individual words in a phrase. A literal parentheses character in the *search_items* must be enclosed with quotes.
"word1 ? word3" "word1 ? ? ? word5"	A question mark indicates that exactly one word is found between the specified words in a phrase. You can specify multiple single question marks, each separated by spaces.
"word1 ?? word6"	A double question mark (with no space between) indicates that from 0 to 6 words are found between the specified words in a phrase.
"word1 [1–3] word5"	Square brackets indicate an interval number of words between the specified words in a phrase: *min-max*. This interval is specified as a variable range, in this case from 1 to 3 missing words.

Semantic index *search_items* can contain the following NLP entity search syntax in addition to the Basic index syntax:

Full Entity and Partial Entity Search:

Argument	Description
{entity}	Specifies the exact wording of a NLP entity. Asterisk wildcards can be applied to individual words in an entity.
<{entity}	A less-than sign prefix specifies an NLP entity ending with the specified word(s). There must be one or more words in the entity appearing before the specified word(s).
{entity}>	A greater-than sign suffix specifies an NLP entity beginning with the specified word(s). There must be one or more words in the entity appearing after the specified word(s).

Multiple *search_items* can be specified, separated by spaces. This is an implicit AND test. For example:

SQL

```
SELECT Narrative FROM Aviation.TestSQLSrch WHERE %ID %FIND
search_index(NarrSemanticIdx,'<{plug electrode} "flight plan" instruct*',0,'en')
```

means that a Narrative text must include one or more SQL Search entities that end with “plug electrode”, AND the positional phrase “flight plan”, AND the word “instruct” with a wildcard suffix, allowing for “instructor”, “instructors”, “instruction”, “instructed”, etc. These items can appear in any order in the text.

3.3.2 Validating an SQL search-items String

You can use the **%iFind.Utils.TestSearchString()** method to validate a *search_items* string. This method enables you to detect syntax errors and ambiguous use of logical operators. For example, `"word1 AND word2 OR word3"` fails validation because it is logically ambiguous. Adding parentheses clarifies this string to either `"word1 AND (word2 OR word3)"` or `"(word1 AND word2) OR word3"`.

The following example invokes this SQL Search utility as an SQL function:

SQL

```
SELECT %iFind.TestSearchString('orange AND (lemon OR lime)')
```

TestSearchString() returns a %Status value: A valid *search_items* string returns a status of 1. An invalid *search_items* string returns an object expression that begins with 0, followed by encoded error information.

3.3.3 Fuzzy Search

InterSystems SQL Search supports *fuzzy search* to match records containing elements (words or entities) that “almost” match the search string. Fuzzy search can be used to account for small variations in writing (color vs. colour), misspellings (collor vs color), and different grammatical forms (color vs. colors).

SQL Search evaluates fuzzy matches by comparing the Levenshtein distance between the two words. The Levenshtein distance is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. The maximum number of single-character edits required is know as the maximum edit distance. The InterSystems SQL Search maximum edit distance defaults to 2 characters. The maximum edit distance is separately applied to each element in the search string. For SQL Search Basic index, it is applied to each word in the search string. For SQL Search Semantic index, it is applied to each NLP entity in the search string. (The examples that follow assume an SQL Search Basic index.)

For example, the phrase “analyse programme behaviour” is a fuzzy search match for “analyze program behavior” when maximum edit distance=2, because each word in the search string differs by an edit distance of (at most) 2 characters: analyse=analyze (1 substitution), programme=program (2 deletions), behaviour=behavior (1 deletion).

A word with that is lesser than or equal to the maximum edit distance is a fuzzy search match for any word with an equal or lesser number of characters. For example, if the edit distance is 2, the word “ab” would match any two-letter word (2 substitutions), any one-letter word (1 substitution, 1 deletion), any three-letter word containing either “a” or “b” (1 substitution, 1 insertion), and any four-letter word containing both “a” and “b” in that order (2 insertions).

- Fuzzy search is supported on all SQL Search index types: Basic, Semantic, and Analytic. On a Basic index it performs fuzzy search on individual words. On a Semantic index it performs fuzzy search on individual NLP entities.
- Fuzzy search cannot be combined with wildcard search.

To activate fuzzy search for **search_index()** specify *search_option* as 3 for fuzzy search with the default edit distance of 2, or 3:n for fuzzy search with an edit distance specified as *n* characters. The following example shows SQL Search with fuzzy search with an edit distance of 4:

SQL

```
SELECT Narrative FROM Aviation.TestSQLSrch WHERE %ID %FIND
search_index(NarrBasicIdx,'"color code" program','3:4','en')
```

Setting 3:1 sets the edit distance=1, which in English is appropriate for matching most (but not all) singular and plural words. Setting 3:0 sets the edit distance=0, which is the same as SQL Search without fuzzy search.

To specify fuzzy search for SQL Search methods, set the pSearchOption = $$$IFSEARCHFUZZY.

3.3.4 Stemming and Decompounding

Basic index, Semantic index, and Analytic index can all support stemming and decompounding. Stemming and decompounding are word-based, not NLP entity-based operations. You must enable stemming and decompounding when you define an SQL Search index. To enable an index for stem-aware searches, specify INDEXOPTION=1; to enable both stem-aware searches and decompounding-aware searches, specify INDEXOPTION=2.

If an SQL Search index was defined to support stemming (1) or stemming and decompounding (2), you can use these facilities in a **search_index()** query by setting the *search_option* value.

3.3.4.1 Stemming

Stemming identifies the Stem Form of each word. The stem form unifies multiple grammatical forms of the same word. When using *search_option*=1 at query time, SQL Search performs search and match operations using the Stem Form of a word, rather than the actual text form. By using *search_option*=0, you can use the same index for regular (non-stemmed) searches.

If stemming is active, search and match is performed by determining the stem form of a search term and using that stem form to match words in the text. For example, the search word `doctors` is a match for either `doctor` or `doctors` in the text. When stemming is active you can match a search word with only its exact match in the text by enclosing a single word in the search list with quotation marks: the search word `"doctors"` is only a match for `doctors` in the text.

3.3.4.2 Decompounding

Decompounding divides up compound words into their constituent words. SQL Search always combines decompounding with stemming; once a word is divided into its constituent parts, each of these parts is automatically stemmed. When using a decompounding search (*search_option*=2), SQL Search compares the decompounding stems of the search term with the decompounded stems of the words in the indexed text field. SQL Search matches a decompounded word only when the stems of *any* of its constituent words match *all* constituent words of the search term.

For example, the search terms “thunder”, “storm”, or “storms” would all match the word “thunderstorms”. However, the search term “thunderstorms” would not match the word “thunder”, because its other constituent word (“storm”) is not matched.

The InterSystems SQL Search decompounding algorithm using a language-specific dictionary that identifies possible constituent words. This dictionary should be populated through the %iKnow.Stemming.DecompoundingUtils class. For example, by pointing it to a text column prior to indexing. You may also wish to exempt specific words from decompounding. You can exempt individual words, character sequences, and training data word lists from decompounding using %iKnow.Stemming.DecompoundUtils.

3.3.5 Languages Not Supported by the InterSystems IRIS Natural Language Processor

You can use SQL Search Basic indexes to index and search texts in languages for which there is no corresponding NLP language model.

Because stemming is not dependent on NLP semantic indexing, you can also perform Basic index word searches on stem forms of words, if a stemmer is available. You must specify INDEXOPTION=1 or INDEXOPTION=2 to perform stem searches. For example, Italian is not an NLP-supported language, but InterSystems IRIS provides a %Text stemmer for Italian.

The following limitations and caveats apply to SQL Search with languages not supported by NLP:

- An InterSystems IRIS Natural Language Processor license is required to use this feature.
- The language must separate words using spaces. Languages that do not uses word separators cannot be searched. However, Japanese (which does not uses word separators) can be searched because NLP provides a Japanese language model.
- Apostrophes do not separate words. NLP recognizes contractions (such as “can’t”) and abbreviated verb forms (such as “there’s”) and separates them into two words, while ignoring apostrophes used for other purposes, such as possession (“John’s”). Without NLP support, SQL Search cannot separate contractions and abbreviations into separate words. You can compensate for this by pre-processing your text, inserting a blank space before or after apostrophes as needed.
- A UserDictionary cannot be applied to the texts before SQL Search indexing.

For further details, refer to Queries Invoking Free-text Search.

3.3.6 Synonym Tables

To implement a synonym table, define the table as a persistent class that extends the iFind.Synonym abstract class.

This class defines two properties, FromTerm and ToTerm. A pair of FromTerm and ToTerm properties define ToTerm as a synonym for FromTerm. SQL Search would use ToTerm to expand the query if the query contains FromTerm.

The query uses the **GetMatch()** method of this class to search the synonyms in the synonym table against the query terms.

During query execution, SQL Search checks if any synonyms exist for a single word unit or a positional search phrase. For example, two synonym pairs ("persons","people") and ("walk","run") are defined in a synonym table. A SQL Search query is executed on the phrase "persons walk". If the synonym table is associated with the query, SQL Search returns not only documents matching the original query, it also returns documents matching any one of the queries: "persons run", "people walk" and "people run".

However, if the *search_items* string is `'"persons walk"'`, query expansion would not happen, because SQL Search does not expand any word in a positional phrase search. The positional phrase itself is the minimum unit for query expansion. If, however, you define a synonym pair like ("persons walk","persons walk and run"), SQL Search would expand the query `'"persons walk"'` to `'"persons walk and run"'`. SQL Search treats a ToTerm as a positional phrase if it contain multiple words. A ToTerm can be any valid positional phrase; it can contain * or ? wildcards.

Note: Synonym tables cannot be used with Regular Expression search (*search_option*=4).

3.4 Highlighting

You can highlight words in a returned text using the *search_items* syntax. Highlighting syntax is:

```
(text,search_items,search_option)
```

search_items: Highlighting uses the same *search_items* syntax as searching. This allows you to use the same *search_items* value for both returning records and highlighting the strings within those records that caused them to be returned. This also allows you to use the **TestSearchString()** method to validate highlighting *search_items* syntax. However, because highlighting is applied to every instance of every match, highlighting ignores the *search_items* syntax AND, OR, and NOT logical operators in a *search_items* string.

search_option: The optional *search_option* can be 0 (the default) or 4 (Regular Expressions).

You can apply highlighting using either of the following:

- SELECT item highlighting:

  ### SQL

  ```
  SELECT %iFind.Highlight(Narrative,'"visibility [1-4] mile*" AND "temp* ? degrees"')
  FROM Aviation.TestSQLSrch
  WHERE %ID %FIND search_index(NarrBasicIdx,'"visibility [1-4] mile*" AND "temp* ? degrees"',0,'en')
  ```

- Utility method highlighting:

  You can use the **%iFind.Utils.Highlight()** method to perform an SQL Search and apply highlighting to the results.

By default, highlighting inserts the <b> and </b> (bold) XML tags at the appropriate places in the string. By default, highlighting is not case sensitive.

Highlighting can be used with any *search_option*, including Regular Expression search (*search_option*=4), as shown in the following example:

ObjectScript

```
SET x="Time flies like an arrow. other stuff. Fruit flies like a banana."
WRITE ##class(%iFind.Utils).Highlight(x,"\p{LU}(\p{L}|\s)+",4)
```

When used with Word Search, this method highlights separately each occurrence of each specified word.

When used with Positional Phrase Search, this method highlights each occurrence of the positional phrase.

3.5 SQL Search Examples

In the following examples, SQL Search Basic index syntax can be used with any type of SQL Search index. SQL Search Semantic index syntax requires a Semantic or Analytic index.

These examples require that you have created and populated the Aviation.TestSQLSrch table, as described in Indexing Sources for SQL Search.

For simplicity of display, these examples return record counts rather than the record text itself. These counts are the number of records that match the search criteria, *not* the number of matches found in the records. A record may contain multiple matches, but is only counted once.

3.5.1 Basic Search Examples

The following examples uses Basic index search to search the Aviation.TestSQLSrch table.

Search for records that contain at least one instance of the words “electrode”, “plug”, and “spark” (in any order):

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'electrode plug spark',0)
```

Note that this is word search, not string search. Therefore, the following example may return different results, and may actually return more results than the previous example:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'electrodes plug spark',0)
```

Search for records that contain at least one instance of a word beginning with “electrode” (electrode, electrodes), and the word phrase “spark plug” (in any order):

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'electrode* "spark plug"',0)
```

Search for records that contain a word beginning with “electrode” (electrode, electrodes), and the word phrase “spark plug” (in any order) within a co-occurrence proximity window of 6 words. Note the punctuation used to specify words and word phrases in a co-occurrence search:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'[electrode*,spark plug,1-6]',0)
```

Search for records that contain the two different word phrases `normal wear` and `"normal" wear`:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'"normal wear"',0)
```

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'"\"normal\" wear"',0)
```

Search for records that contain at least one word containing the string seal (seal, seals, unseal, sealant, sealed, previously-sealed), and the word phrase “spark plug”:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'*seal* "spark plug"',0)
```

Search for records that contain the wildcard phrase “wind from ? ? at ? knots.” Possible values might include “wind from the south at 25 knots” and “wind from 300 degrees at ten knots.” Note that if there is a space between two sequential question marks (`? ?`) the wildcard represents exactly two words; if there is no space between the two question marks (`??`) the wildcard represents from 0 to 6 words:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'"wind from ? ? at ? knots"',0)
```

The following example uses Basic index with Regular Expression search (with *n*=4). It searches records that contain occurrences of strings specifying dates between “January 10” and “January 29” inclusive:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrBasicIdx,'"January [1-2][0-9]"',4)
```

For further details, refer to Regular Expressions.

3.5.2 Semantic Search Examples

The following examples use Semantic index search to search the Aviation.TestSQLSrch table.

Search for records that contain the NLP entity “spark plug electrodes”:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrSemanticIdx,'{spark plug electrodes}',0)
```

Search for records that contain an NLP entity ending with “spark plug” or “spark plugs”:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrSemanticIdx,'<{spark plug*}',0)
```

Search for records that contain both an NLP entity ending with “spark plugs” and the NLP entity “spark plugs”:

SQL

```
SELECT COUNT(Narrative) FROM Aviation.TestSQLSrch
WHERE %ID %FIND search_index(NarrSemanticIdx,'<{spark plugs} {spark plugs}',0)
```

4

SQL Search REST Interface

This topic describes the REST interface for InterSystems SQL Search.

4.1 REST Syntax for SQL Search

SQL Search provides the REST API to access SQL Search indexed results.

The endpoint syntax is `"/table/:TableName/search"` where TableName is a table which contains at least one SQL Search index.

A sample access path would be like `http://localhost:443/api/iKnow/v1/user/table/iFind.Table/search`.

You need to use "POST" to access the endpoint. You can put the search parameters as a JSON object in the body of the request.

4.2 Search Parameters

The following are the supported parameters:

```
{       "query":"string",
   "index":"string",
   "option": 0,
   "distance": "string",
   "language":"string",
   "includeText": 0,
   "columns": ["string"],
   "highlightSpec": {
      "tag": "<b>",
      "limit": 0,
      "name": "Highlighted"
      },
   "rankSpec": {
      "name": "Rank"
     }
 }
```

`"query"` specifies the SQL Search query.

`"index"` specifies which SQL Search index you would like to search. If you do not specify an index, SQL Search uses the first index found.

`"option"` is an integer that specifies the type of search to perform. Available values include 0 (syntax search), 1 (syntax search with stemming), 2 (syntax search with decompounding and stemming), 3 (syntax search with fuzzy search), and 4 (syntax search with Regular Expressions).

`"language"` specifies a two-character string that specifies the language of the text. For example, `"en"` specifies English.

`"includeText"` and `"columns"` specify which columns you would like to return. When `"includeText"` is specified as 1, the field indexed by the SQL Search index is returned. You can use `"columns"` to specify other column names as a JSON array of strings.

`"highlightSpec {name}"` specifies the column alias name for the highlight column.

`"rankSpec {name}"` specifies the column alias name for the rank column.

The returned results format is a JSON object. The returned rows are formatted in a "rows" JSON array: `{ "rows":[{},{},{}] }`.