# First Look: SQL Search with InterSystems Products

Version 2019.4
2020-01-28

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

*First Look: SQL Search with InterSystems Products*
InterSystems IRIS Data Platform   Version 2019.4   2020-01-28

# Table of Contents

# First Look: SQL Search with InterSystems Products

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

- Why SQL Search Is Important
- How InterSystems IRIS Implements SQL Search
- Trying SQL Search for Yourself
- For More Information about SQL Search

This First Look presents an introduction to SQL context-aware text searching and walks through some initial tasks associated with indexing text data for searching and performing SQL Search. Once you've completed this exploration, you will have indexed text in an SQL column for text searching and performed several types of searches. These activities are designed to use only the default settings and features, so that you can acquaint yourself with the fundamentals of the feature. For the full documentation on SQL Search, see the SQL Search Guide.

A related, but separate, tool for handling unstructured texts is Natural Language Processing (NLP). SQL Search presupposes that you know what you are looking for. NLP text analysis allows you to analyze the contents of texts with no prior knowledge of the text contents.

To browse all of the First Looks, including those that can be performed on a free evaluation instance of InterSystems IRIS, see InterSystems First Looks.

## 1 Why SQL Search Is Important

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 How InterSystems IRIS Implements SQL Search

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 indices, 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.

# 3 Trying SQL Search for Yourself

It’s easy to use InterSystems IRIS SQL Search. This simple procedure walks you through the basic steps of searching text data stored as a string in an SQL table column.

To use the procedure, you will need a running InterSystems IRIS instance. Your choices for InterSystems IRIS include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see Deploying InterSystems IRIS in *InterSystems IRIS Basics: Connecting an IDE*.

You will also use Atelier, the Eclipse-based IDE for InterSystems IRIS, to create ObjectScript code in your instance. For instructions for setting up Atelier and connecting it to your instance, see Atelier in *InterSystems IRIS Basics: Connecting an IDE*; more detailed information is available in *First Look: Atelier with InterSystems Products*. (You can also use the Studio IDE from InterSystems, a client application running on Windows systems, to create the ObjectScript code; for more information, see *Using Studio* and Studio in *Connecting an IDE*.)

## 3.1 Before You Begin

To use the procedure, you will need a running InterSystems IRIS instance. Your choices for InterSystems IRIS include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see Deploying InterSystems IRIS in *InterSystems IRIS Basics: Connecting an IDE*.

You will also use Atelier, the Eclipse-based IDE for InterSystems IRIS, to create ObjectScript code in your instance. For instructions for setting up Atelier and connecting it to your instance, see Atelier in *InterSystems IRIS Basics: Connecting an IDE*; more detailed information is available in *First Look: Atelier with InterSystems Products*. (You can also use the Studio IDE from InterSystems, a client application running on Windows systems, to create the ObjectScript code; for more information, see *Using Studio* and Studio in *Connecting an IDE*.)

Finally, you will need to obtain the Aviation.Event table and associated files from the GitHub repo https://github.com/intersystems/Samples-Aviation.

## 3.2 Downloading and Setting up the Sample Files

The Samples-Aviation sources must be accessible by the instance. The procedure for downloading the files depends on the type of instance you are using, as follows:

- If you are using an ICM-deployed instance:
  1. Use the **icm ssh** command with the **-machine** and **-interactive** options to open your default shell on the node hosting the instance, for example:

     ```
     icm ssh -machine MYIRIS-AM-TEST-0004 -interactive
     ```

  2. On the Linux command line, use one of the following commands to clone the repo to the data storage volume for the instance. For a configuration deployed on Azure, for example, the default mount point for the data volume is /dev/sdd, so you would use commands like the following:

     ```
     $ git clone https://github.com/intersystems/FirstLook-SQLBasics /dev/sdd/FirstLook-SQLBasics
     OR
     $ wget -qO- https://github.com/intersystems/FirstLook-SQLBasics/archive/master.tar.gz | tar xvz
      -C /dev/sdd
     ```

  The files are now available to InterSystems IRIS in /irissys/data/FirstLook-SQLBasics on the container's file system.

- If you are using a containerized instance (licensed or Community Edition) that you deployed by other means:
  1. Open a Linux command line on the host. (If you are using Community Edition on a cloud node, connect to the node using SSH, as described in *Getting Started with InterSystems IRIS Community Edition*.)
  2. On the Linux command line, use either the **git clone** or the **wget** command, as described above, to clone the repo to a storage location that is mounted as a volume in the container.
     - For a Community Edition instance, you can clone to the instance's durable %SYS directory (where instance-specific configuration data is stored). On the Linux file system, this directory is /opt/ISC/dur. This makes the files available to InterSystems IRIS in /ISC/dur/FirstLook-SQLBasics on the container's file system.
     - For a licensed containerized instance, choose any storage location that is mounted as a volume in the container (including the durable %SYS directory if you use it). For example, if your **docker run** command included the option **-v /home/user1:/external**, and you clone the repo to /home/user1, the files are available to InterSystems IRIS in /external/FirstLook-SQLBasics on the container's file system.

- If you are using an InterSystems Learning Labs instance:

1. Open the command-line terminal in the integrated IDE.
2. Change directories to /home/project/shared and use the **git clone** command to clone the repo:

   ```
   $ git clone https://github.com/intersystems/FirstLook-SQLBasics
   ```

   The folder is added to the Explorer panel on the left under **Shared**, and the directory is available to InterSystems IRIS in /home/project/shared.

- If you are using an installed instance:
  - If the instance's host is a Windows system with GitHub Desktop installed:
    1. Go to https://github.com/intersystems/Samples-Aviation in a web browser on the host.
    2. Select **Clone or download** and then choose **Open in Desktop**.

    The files are available to InterSystems IRIS in your GitHub directory, for example in C:\Users\User1\Documents\GitHub\FirstLook-SQLBasics.
  - If the host is a Linux system, simply use the **git clone** command or the **wget** command on the Linux command line to clone the repo to the location of your choice.

Once you have the sample files, follow the steps provided in the Samples-Aviation README.md file under "Setup instructions":

1. Create a namespace called **SAMPLES** as follows:
   a. Open the Management Portal for your instance in your browser, using the URL described for your instance in *InterSystems IRIS Basics: Connecting an IDE*.
   b. Select **System Administration** > **Configuration** > **System Configuration** > **Namespaces** to go to the **Namespaces** page.
   c. On the **Namespaces** page, select **Create New Namespace**. This displays the **New Namespace** page; follow the instructions for using this page in Create/Modify a Namespace in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*. Call the new namespace **SAMPLES**.
   d. Select **Save** near the top of the page and then select **Close** at the end of the resulting log.

2. To enable the SAMPLES web application for use with InterSystems IRIS Analytics:
   - a. In the Management Portal, click **System Administration > Security > Applications > Web Applications**.
   - b. Click the **/csp/samples** link in the leftmost column (assuming that the namespace you created is called **SAMPLES**).
   - c. In the **Enable** section, select **Analytics**.
   - d. Click **Save**.

3. Open the InterSystems IRIS Terminal using the procedure described for your instance in *InterSystems IRIS Basics: Connecting an IDE* and enter the following command to change to the namespace where the sample will be loaded:

   ```
   set $namesapce="SAMPLES"
   ```

4. Enter the following command, replacing *.path* with the full path of the directory that contains the README.md and LICENSE files of the repo you cloned or downloaded:

   ```
   do $system.OBJ.Load("<path>\buildsample\Build.AviationSample.cls","ck")
   ```

5. Enter the following command:

   ```
   do ##class(Build.AviationSample).Build()
   ```

When prompted, enter the full path of the directory that contains the README.md and LICENSE files. The method then loads and compiles the code and performs other needed setup steps.

## 3.3 Creating and Testing a Basic SQL Search Index

Once the code is compiled, which may take a minute or two, continue with the following steps:

1. Open Atelier, verify that it is connected to your instance, and create a Basic SQL Search index by defining the following class in the **SAMPLES** namespace:

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

   This example creates a persistent class (table) that contains a Narrative property (column), and defines a Basic SQL Search index for this property. Because this is a new class, you must populate the table with text data.

2. Populate the table with text data and build the SQL Search index. An SQL Search index is built and maintained like any other SQL index.

   and enter the following commands to populate the new table with text data from the Aviation.Event table you downloaded. In this example, the SQL Search index is automatically built as each record is added:

   ```
   ZNSPACE "SAMPLES"
   SET in1="INSERT OR UPDATE INTO Aviation.TestSQLSrch (UniqueNum,Narrative) "
   SET in2="SELECT %ID,NarrativeFull FROM Aviation.Event WHERE %ID < 100"
   SET myinsert=in1_in2
   SET tStatement=##class(%SQL.Statement).%New()
   SET qStatus=tStatement.%Prepare(myinsert)
     IF qStatus'=1 {WRITE "%Prepare failed:" DO $System.Status.DisplayError(qStatus) QUIT}
   SET rset=tStatement.%Execute()
   WRITE !,"Total rows inserted=",rset.%ROWCOUNT
   ```

   For performance reasons, you may wish to use the %NOINDEX option to defer building indices until the table is fully populated, and then build the SQL Search index (and any other defined indices) using the %Build() method.

   Alternatively, you could add an SQL Search index to an existing persistent class that already contains text data, and then populate the SQL Search index using the %Build() method.

3. Open a SQL Shell in Terminal, as described in the first few steps of Creating and Populating a Table With a SQL Script File in *First Look: InterSystems SQL*, and use SQL Search as a WHERE clause condition of a **SELECT** query. The WHERE clause can contain other conditions associated by AND logic. Run the following SQL Query in the SAMPLES namespace:

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

   - The **search_index()** function specifies a *search_index* parameter. This is a defined SQL Search index for the property (column) to be search. It can be a Basic, Semantic, or Analytic index.

   - The **search_index()** function specifies a *search_item* parameter.

     This example defined the *search_item* as `"visibility [1-4] mile*" "temperature ? degree*"`. This returns all records that contain both positional phrases, in any order:

`"visibility [1-4] mile*"` returns phrases with from 1 to 4 words between the words “visibility” and “mile”. Because mile* specifies a wildcard, it could match either mile or miles. For example, “visibility less than 1 mile”, “visibility 10 miles”, “visibility approximately 20 statute miles”, “visibility for many miles”.

`"temp* ? degrees"` returns phrases with a word beginning with “temp” and ending in 0 or more non-space wildcard characters, a single missing word, and then the word “degrees.” Thus it would return records with the phrase “temperature 20 degrees”, “temp. 20 degrees”, “temperature in degrees”, and also the (probably unintended) “temporarily without degrees”.

- The **search_index()** function can optionally specify a *search_option* parameter.
- This option can apply an optional transformation to the search, as follows: 1=stemmed search applies a stemmer to match words or phrases based on their stem form. 2=decompounding search applies decompounding to compound words. 3=fuzzy search applies a specified degree of fuzziness (number of character differences) to the search. 4=regular expression search allows searching using RegEx matching. This example specifies the default, 0, meaning no search transformation.
- The **search_index()** function can optionally specify a *search_language* parameter. You can specify a language, or specify `'*'` to invoke automatic language identification, supporting searching texts containing multiple languages. This example specifies the default, `'en'` (English).

This example also highlights the returned text by applying the same *search_item* to the returned records. This highlights every instance of either of these phrases by delimiting them with **<b>** and **</b>** tags.

This example is provided to give you some initial experience with InterSystems IRIS SQL Search. You should not use this example as the basis for developing a real application. To use SQL Search in a real situation you should fully research the available choices provided by the software, then develop your application to create robust and efficient code.

# 4 Learn More About SQL Search

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

- Using InterSystems SQL Search