



## **ITRS Insights v1.4**

**Query, Search and Transform User Guide v1.0**

Copyright 2016. ITRS Group Ltd. All rights reserved.  
Information in this document is subject to change without notice. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of those agreements. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of ITRS Group Ltd.

ITRS Group Ltd  
6th Floor, The Bonhill Building, 15 Bonhill Street,  
London, EC2A 4DN, UK  
t: +44 (0)20 7638 6700  
f: +44 (0)20 7256 5760

# CONTENTS

## CHAPTER 1

ITRS Insights Query, Search and Transform Introduction .....	6
Insights Query .....	13
Insights FREE TEXT Search .....	26
Transform Data .....	28

## CHAPTER 2

Organizing Data in ITRS Insights .....	31
Introduction .....	32
Cities are getting smarter .....	33
Stream .....	33
Contributor .....	36
Domain .....	37
Tenant .....	39
Collection .....	40

## GLOSSARY

## APPENDIX

Query Clauses .....	46
Query Functions .....	47
abs: absolute value .....	55
contains .....	55
counter .....	56
EMA .....	56
Parameters .....	56
first .....	56
format .....	57
Form: .....	57
geoCircle .....	58
geoPoint .....	58
geoRectangle .....	58
Last .....	59
max .....	60
Max .....	62
min .....	63
Min .....	64
prefix .....	65
sum .....	66
Query Operators .....	68

**This page intentionally left blank.**

# CHAPTER 1

# ITRS INSIGHTS QUERY, SEARCH AND TRANSFORM INTRODUCTION

---

ITRS Insights provides you with two powerful tools for analysing and searching your data:

**Free text SEARCH:** a string search for historical data stored in semi-structured repositories (SSR)

**Query:** a powerful SQL-like query used for historical and streaming data stored in any repository type (semi-structured repository or time-series repository)

This user guide talks you through how Insights uses **queries** and **free text searches**, how to create basic and sophisticated queries, and how to run free text searches.

If you're an advanced user and you're tempted to skip the introduction, please read our section on "The Transformational Pipeline" on page 14, as this will likely be different from your previous SQL experience.

Once you've run your query, you'll almost certainly want to visualise the results. For this, check out our [Insights visualising Results User Guide](#).

## Introduction

### *The difference between search and query*

Insights' **Free Text Search** looks through your data streams and searches for a specific string of text that you define in the search bar (characters, numbers etc.). It returns to you all the instances where this bit of text occurs. Free text search does not attempt in anyway to interpret or know the data - it is, in effect, simply a matching exercise.

A query seeks to in some way understand and work with the data. For example, in its most basic form a query can look into data to find instances of a number that is greater than 5. To do this requires that the system understands what a number is (and what it is not) and the comparative order of the numbers - this is why you can only run queries on data that has a **schema**

This example of a query represents the most simplistic of queries: through that manipulation and comparison of the data, you can process and analyse your data stream in any number of ways. Insights provides you not only with basic manipulation functions, but with easy and quick access to complex and sophisticated algorithms for things such as anomaly detection and univariate analysis.

### *Repository Considerations*

If your streams are live, you can skip this section. If you're storing your data, then read on.

The type of search you can run is determined by the type of **repository** your data is stored in. In short, you can only run free text searches on **semi-structured repositories**.

A **time series repository** gives Insights an understanding, to some extent, of the concepts of time. It also comes with some complex algorithms built into the query language. Combined, these mean that it's possible to run a single query that is sophisticated enough to spot anomalies in the data.



---

For instance, if aging Mrs Brown at No 4 always watches her favorite soap opera at 7pm every evening and straight afterward gets up and makes a cup of tea, then the notebook can help you spot that this evening she didn't switch the kettle on at 8pm, which may be cause for concern. Because it is real-time then the queries can immediately identify anomalies, however you can also run the same thing against historic data to find previous patterns and areas for further investigation.

If you have yet to import a data stream into Insights, or if you need help figuring out which repository type is best for you, check out our [Importing Data Streams & Schema User Guide](#).

### *Notebooks*

Your searches and queries are saved in **Notebooks**, and these in turn can be categorized by the **collections** they belong to. This means that you can use Collections for high-level organizational teams or functional requirements, then create subsets of Notebooks for the different analytics that you want to run.

What's more, you can import and export your Notebooks, so you can share them with your colleagues or transfer them between Insights installations.

For more on **notebooks** and **collections**, see our [Dashboard & Notebook User Guide](#).

### *Viewing the Results*

Once you've run your searches or queries, your results are displayed on a **dashboard**.

If you haven't configured any charts, the results are tabulated numerically, but you can easily select one or more of our pre-built charts to visualise the results. See: [Visualising Results User Guide](#).


---

## Before you Start

Before you read further into this user guide, make sure you have:


1. Valo running...

### How to Launch Valo

1. Navigate to the `\valo\bin` directory in your Valo installation folder
2. Double click **valo** 

2. Insights running...

### How to Launch Insights

1. Navigate to the Insights installation folder
  2. Double click **ITRS-Insights** 
3. ...a data stream in an SSR or TSR

Ask your systems administrator if you need help with this bit.

# Query and Free Text Search Interface



1. **Active tab** - identifies the Domain, Dashboard, Notebook or Collection that is being displayed
2. **Re-ordering bar** - drag and drop to reorder your query or free text search
3. **Breadcrumbs** - provides a visual indication of which Collection you are working in
4. **Query Panel** - a text box in which you enter your query. Insights features a live syntax highlighter to enable you to rapidly see any errors in your query syntax
5. **Free Text Panel** - a text box in which you enter your text string
6. **Run** - runs all of the queries and free text searches in the active Notebook
7. **Edit** - allows you to edit the title and description
8. **Export** - opens a dialog to allow you to save the active component to a location of your choice
9. **Run Query** - runs the query
10. **Exposed Parameter** - entry fields for parameters exposed in the Query Panel
11. **Time Boundary Selection** - visible only in free text searches, these controls allow you to specify a time and date range for returned results

12. **Additional Options** - opens a dialog displaying additional options (duplicate and delete)
13. **Search** - runs the free text search
14. **Add** - adds a new query, free text search or text comment

For information about the interface displayed once a free text search or query has been run.

ITRS Insights provides you with a simple SQL-like query language that enables you to get the most from your data streams.

This guide will talk you through the principles that underpin querying in Insights, explain how to create a simple query from scratch, and provide reference material for more experienced users to use to build sophisticated queries.

### Query Basics

Insights makes use of a simple SQL-like query language. This means that we use a simple syntax that will be familiar to anyone with any SQL experience.

Insights features a live syntax highlighter to enable you to rapidly see any errors in your query syntax.

As you type your query, valid clauses, functions and operators turn **blue** once you've typed them

To run a basic query, you only need three simple clauses: `from`, `select` and `where`.

#### *from*

**from:** specifies which stream will be used as the initial source for data in the query.

`from` is used in every single Insights query, followed by the entity that you are querying:

```
from <stream_name>
```

If you are querying historical data (in a TSR), then you also have to use the operator `historical` in front of the stream name. For example:

```
from historical <stream_name>
```

The most obvious difference from regular SQL is that in Insights queries, `Select` comes after `from`. There is a good reason for this: "The Transformational Pipeline" below.

### *where*

**where** : a clause used to return only rows that match a specified condition. Field identifiers, scalar functions, or operators can be also used in a where clause.

**Where** allows you to further transform your query by specifying a condition to query.

```
from historical <stream_name>
where value > 50
```

### *select*

**select**: used to specify which fields should appear in the results, Multiple fields are separated by commas. Fields in a select clause can be field identifiers, scalar functions, or values computed with operators.

**Select** is used to determine which fields you want the query to return.

```
from historical <stream_name>
where value > 50
select field1, field2, field3
```

## The Transformational Pipeline

**Transformational Pipeline**: each query instruction takes an input and produces an output via some form of transformation.

Insights queries works as a pipeline of stream transformations.

This is best explained using a simple example. Let's assume we have a stream `/streams/demo/infrastructure/cpu` that is stored in a repository containing the following data:

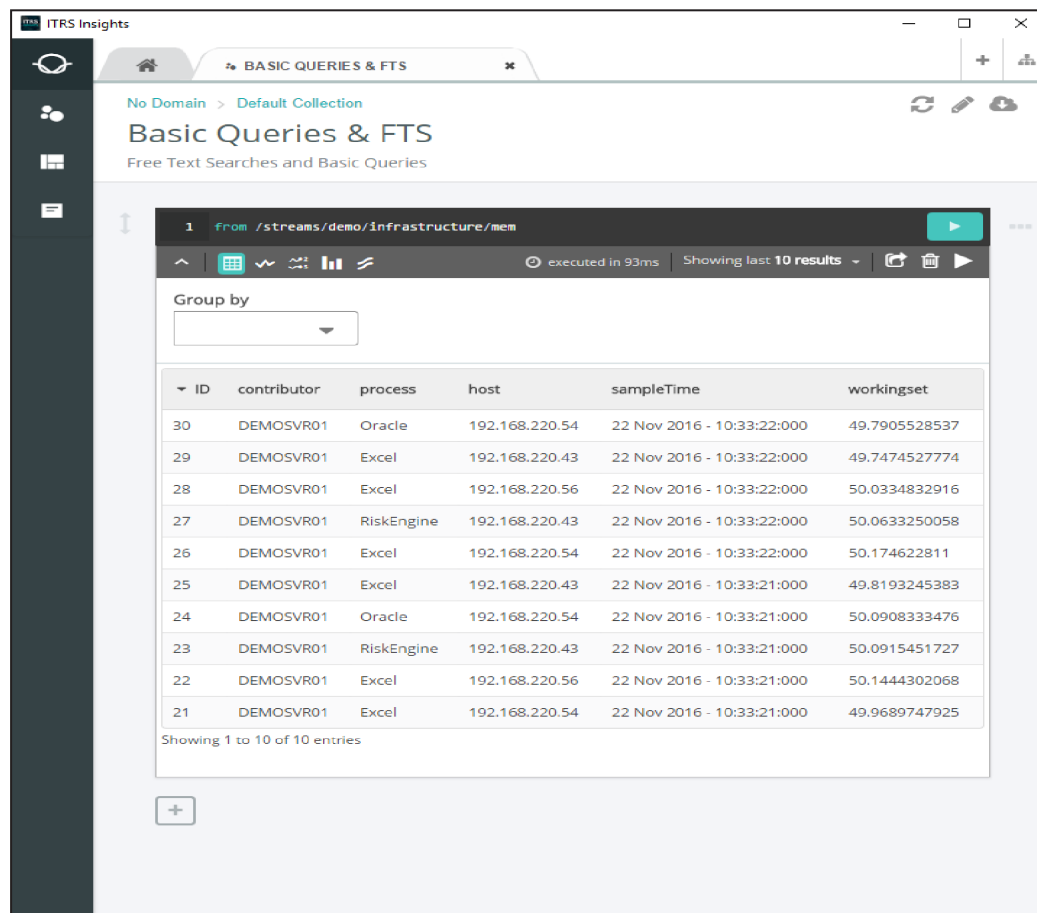
[

```
{ "host": "c.itrs", "process": "Excel", "user":
50.0, "kernel": 75.0 },
{ "host": "a.itrs", "process": "Oracle", "user":
60.0, "kernel": 25.0 },
{ "host": "b.itrs", "process": "RiskEngine", "user":
70.0, "kernel": 99.0 }
]
```

We start by just getting the data from the repository. To do this, we simply use a `from` query with the `historical` key word, but you can perform this on streaming data, too:

```
from historical /streams/demo/infrastructure/cpu
```

This will give us the data from the stream without any transformations being applied:



The screenshot shows the ITRS Insights web application. The main heading is "Basic Queries & FTS" with the subtitle "Free Text Searches and Basic Queries". Below this, a query is entered in the top bar: `1 from /streams/demo/infrastructure/mem`. The query has been executed in 93ms and is showing the last 10 results. A "Group by" dropdown is visible above the table. The table itself has columns: ID, contributor, process, host, sampleTime, and workingset. It displays 10 rows of data, with the last row (ID 21) highlighted. Below the table, it says "Showing 1 to 10 of 10 entries".

ID	contributor	process	host	sampleTime	workingset
30	DEMOSVR01	Oracle	192.168.220.54	22 Nov 2016 - 10:33:22:000	49.7905528537
29	DEMOSVR01	Excel	192.168.220.43	22 Nov 2016 - 10:33:22:000	49.7474527774
28	DEMOSVR01	Excel	192.168.220.56	22 Nov 2016 - 10:33:22:000	50.0334832916
27	DEMOSVR01	RiskEngine	192.168.220.43	22 Nov 2016 - 10:33:22:000	50.0633250058
26	DEMOSVR01	Excel	192.168.220.54	22 Nov 2016 - 10:33:22:000	50.174622811
25	DEMOSVR01	Excel	192.168.220.43	22 Nov 2016 - 10:33:21:000	49.8193245383
24	DEMOSVR01	Oracle	192.168.220.54	22 Nov 2016 - 10:33:21:000	50.0908333476
23	DEMOSVR01	RiskEngine	192.168.220.43	22 Nov 2016 - 10:33:21:000	50.0915451727
22	DEMOSVR01	Excel	192.168.220.56	22 Nov 2016 - 10:33:21:000	50.1444302068
21	DEMOSVR01	Excel	192.168.220.54	22 Nov 2016 - 10:33:21:000	49.9689747925

Now we can modify the query and start performing some transformations.

Lets add a `where` clause:

---

```
from historical /streams/demo/infrastructure/cpu
where kernel > 10
```

The results of this first transformation would be:

```
[
  { "host": "c.itrs", "process": "Excel", "user":
    50.0, "kernel": 75.0 },
  { "host": "b.itrs", "process": "RiskEngine", "user":
    70.0, "kernel": 99.0 }
]
```

Lets add a further transformation, ordering by the host field:

```
from historical /streams/demo/infrastructure/cpu
where kernel > 50
order by host
```

Now we have:

```
[
  { "host": "b.itrs", "process": "RiskEngine", "user":
    70.0, "kernel": 99.0 },
  { "host": "c.itrs", "process": "Excel", "user":
    50.0, "kernel": 75.0 }
]
```

Our final transformation will select some fields and alias them:

```
from historical /streams/demo/infrastructure/cpu
where kernel > 50
order by host
select process as p, user as u, kernel as k
```

This returns:

```
[
  { "p": "RiskEngine", "u": 70.0, "k": 99.0 },
  { "p": "Excel", "u": 50.0, "k": 75.0 }
]
```

As each 'instruction' operates on the data emitted from the previous instruction, transformations can be specified in any order, the following query could be re-written like so and it would emit the same results:

```
from historical /streams/demo/infrastructure/cpu
select process as p, user as u, kernel as k
order by host
where kernel > 50
```

## Exposed Parameters

To better support front-end users who may use queries but not actually write

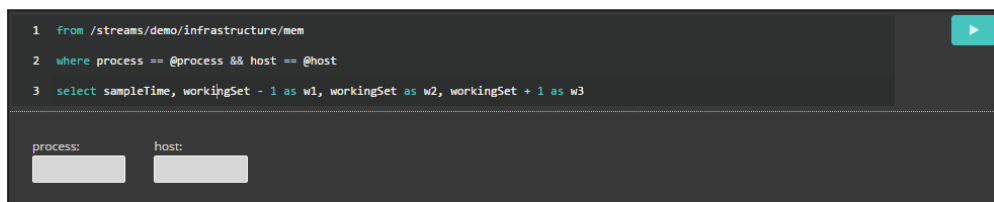


them, one or more parameters in a query can be exposed.

Exposing a parameter in a query is achieved by using @, followed by the parameter name.

In the following example, parameter windows for `process` and `host` will appear beneath the query (this happens in real-time, as the parameters are exposed):

```
from historical /streams/demo/infrastructure/mem
where process == @process && host = @host
select sampleTime, workingSet - 1 as w1, workingSet as
w2, workingSet = 1 as w3
```






Once the query has been written, end-users can enter strings into the parameter windows without having to make any changes to the SQL-like code.

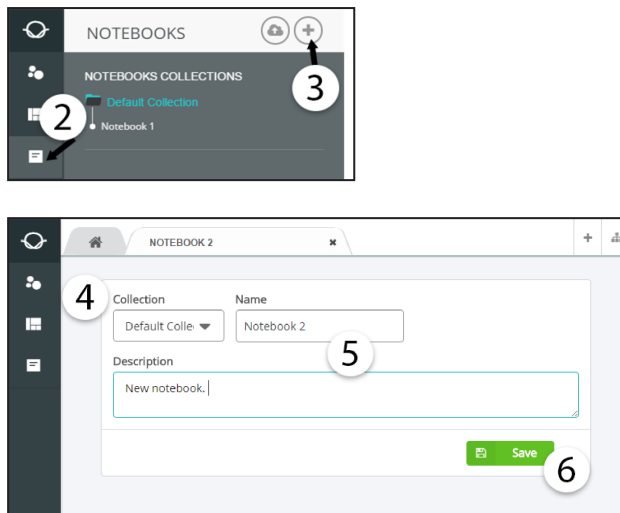
## Run a Query

### Select a Notebook

To run a query in Insights, you need a **Notebook**. If you don't already have one, create one:

#### How to Create a Notebook





1. Run **ITRS Insights**
2. Click **Notebook**  on the main navigation panel
3. Click **add**  - a new Notebook tab will open
4. Select a **Collection** from the **drop-down dialog**, or click inside the text field and type the name of the **Collection** that you want to create
5. Enter a **name** and **description** for the new Notebook
6. Click **Save** 

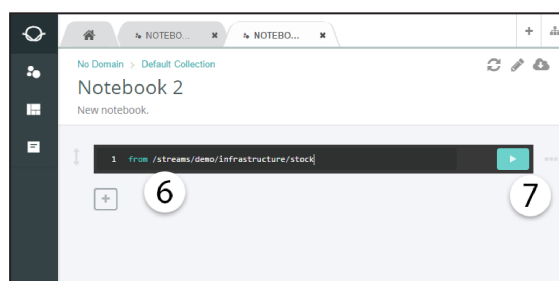
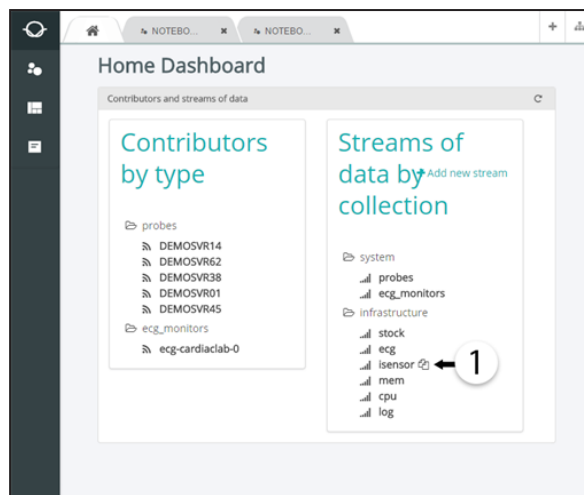


Once you've created a notebook, you're ready to write and run queries.

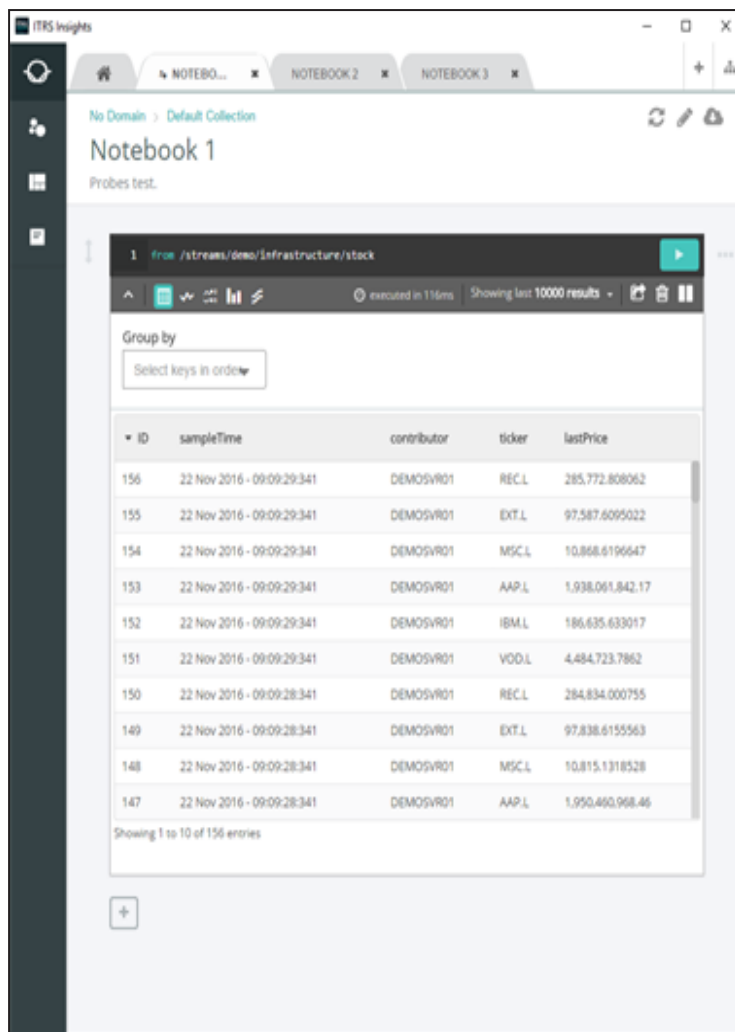
Start with a simple `from` or `from historical query` - it'll allow you see your data without applying any filter or analytics to it.

### How to Create a Basic Query

1. (Optional) Hover your cursor over the **Stream** of interest, then click the **Copy to Clipboard** . This will commit a basic `from` query of that stream to the clipboard
2. On the main navigation panel, click **Notebook** 
3. From the tree, select the **Notebook** to save the query in
4. Click **Add Query/Search** 
5. Click **Query**
6. Press CTRL V on your keyboard to paste the `from` query into the **query field**, or **manually enter** the stream you wish to query
7. Click **Run** 



If you ran your query on a valid stream using valid syntax, you should now see your data streaming into the results window in a table format:




You can now group your results and sort them by clicking on the column headers in the same way that you would a commercial spreadsheet program.

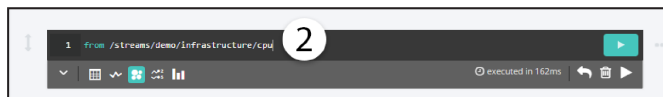
You'll also notice that the visualization panel has appeared below the query you wrote. This allows you to start visualising your data, and even start putting it into **Dashboards**.

The query you created will be saved automatically. You can edit it, duplicate it or delete it at any time.




---

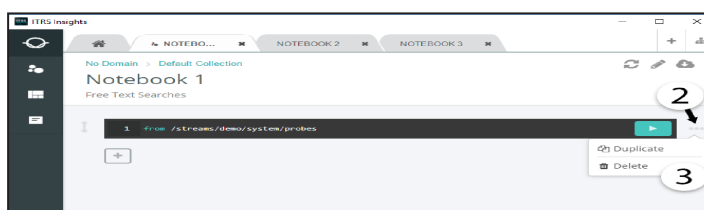
## How to Edit a Query

1. Click **Notebook**  and select the Notebook that contains the query
2. Click the mouse cursor in the **query field**, then make your edits
3. Run your query or navigate to another part of the interface - **your edits will be saved automatically**



## How to Delete a Query

1. Click **Notebook**  and select the Notebook that contains the query
2. Click **breadcrumbs** 
3. Select **delete**  from the drop-down menu



## Run an Advanced Query

Now that you've run a simple from query, it's time to discuss how you can build more complex queries.

We're assuming that you have read through "The Transformational Pipeline" on page 14 and understand the transformational nature of the Insights query, but if you have not then do so now.

Advanced querying of your data streams requires you to understand the three key components:

1. What **format** your data is in, and the **fields** it contains (is the stream made up of dates, time, geocoordinates, ISO numbers? Does it contain strings, Boolean values?) While this information is defined by the schema that accompanied the stream when it was imported into Insights, you've got to be familiar with it if you want to query it
2. How to combine SQL-like **clauses** (sometimes called **transformations**), **functions** and **operators** together to interrogate your data
3. How you can use Insights' concepts - **Domains** and **Collections**, for example - to enhance your ability to get the most from your data


### *Your Data Stream*

If you are familiar with your stream, move on to learning about the query syntax and constructions that you can use in Insights.

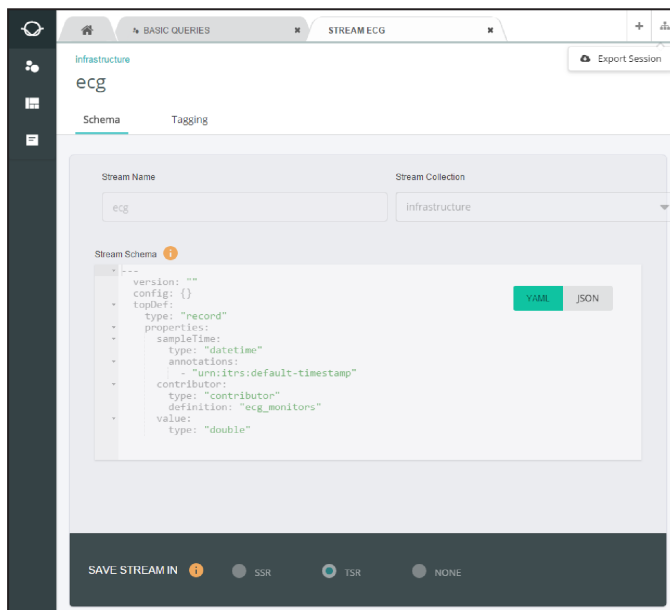
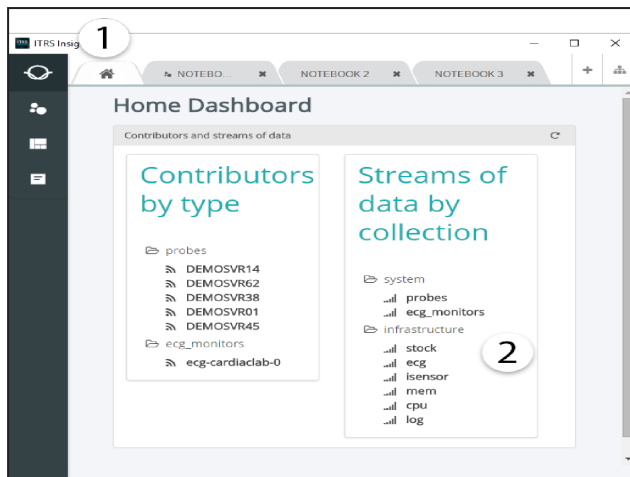
If you don't know what data your stream contains, take a look at the **schema** for the stream.

**Schema:** a document of understanding that defines what data a stream contains - its structure and shape - and in what format.

### How to View a Schema

1. Click **Home** 
2. In the **Streams by Collection** window, Click the **stream** that you want to inspect

A new tab will open displaying the schema for that stream.



## Insights' Query Syntax & Constructions

Query clauses are the foundation of Insights' transformational query capability.

There are a limited number of them, and if you have any SQL experience you should be familiar with them. If not, we'd recommend taking a free online SQL course as a quick primer (you can try [Code Academy](#), for example).

Combine the transformational clauses with "Query Functions" on page 47 and "Query Operators" on page 68 to deliver sophisticated query capabilities.

There are a huge range of functions to work with in Insights. These can be divided into two groups: scalar and aggregate.

### *Scalar*

Scalar functions operate on a single input value and returns an output value. For numeric data, these include simple arithmetic operations (addition, subtraction, multiplication and division). For string values, examples of scalar functions include transforming a string into lower or upper case or even to a datetime format.

Let's say we want to create a new field for the hour of day, we can use Insights' hour function to convert the `sampleTime` field into a numeric field:

```
from /streams/environment/sensors/air
select hour(sampleTime) as hourOfDay, area, carbonm
```

Note how we have also given our new field an alias of 'hourOfDay' using the term 'as'.

### *Aggregate*

Aggregate functions operate on a group of input values to compute a single result. Examples include count, "counter" on [page 56](#), "sum" on [page 66](#), avg, "min" on [page 63](#) and "max" on [page 60](#).

You can use them on historical data using the `historical` clause, or you can use them on real-time streaming data.

In real time queries, in order to use aggregate functions, we firstly need to group by a time dimension (using a window), followed by an aggregation function. We can also group by further variables after the time window.

For example:

```
from /streams/environment/sensors/air
where carbonm > 15
group by sampleTime window of 2 minutes, area
select sampleTime, area, count(carbonm) as
criticalCarbonMCount
```



---

## *Using Insights' Data Structure to Improve your Queries*

**Domains** can be used to filter the results of a **query** to only the data emanating from **contributors** in the Domain. This allows you to contextualise and re-use a particular query across different sources without having to modify it.

For instance, you can set up a domain that is all Windows servers in a particular IP range, including only data with tags `critical` and `error`. In this way you can run simple queries on this Domain, knowing that the appropriate data (i.e. `critical` and `error`) is always dynamically included in real-time.

See Query Functions and Query Operators for more information

## INSIGHTS FREE TEXT SEARCH

Insights' allows you to conduct a free text search across all of your data streams at the same time.



In Insights v1.4, only a single search term can be entered, and Boolean operators are not supported.

Similar to an internet search engine, **Free Text Search** searches your data for a number or text string that you specify.

The search capability supports the wildcard \*, allows you to search within a query and across multiple data streams. It returns and orders results based on their relevance.

### Run a Free Text Search


#### How to Create a Free Text Search

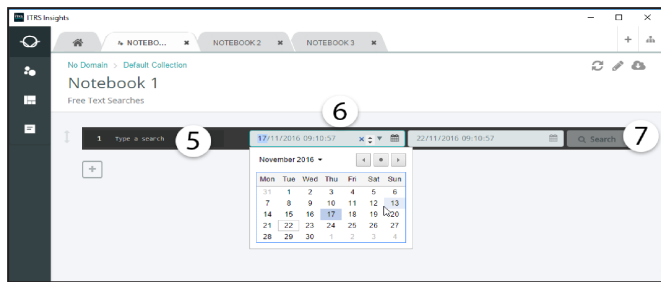
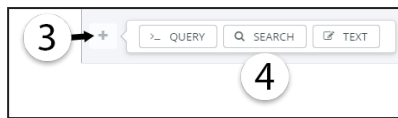
1. On the main navigation panel, click **Notebook** 
2. From the tree, select the **Notebook** to run the search from
3. Click **Add Query/Search** 
4. Click **Search**
5. Enter your **search string**

Note: use double quotes around your search string: "your text here".

Note: You can use the wildcard \* in your free text search, but the search must not start with it.

Note: string searches are case sensitive.

6. (optional) enter **date** or **time search boundaries**
7. Click **Run** 

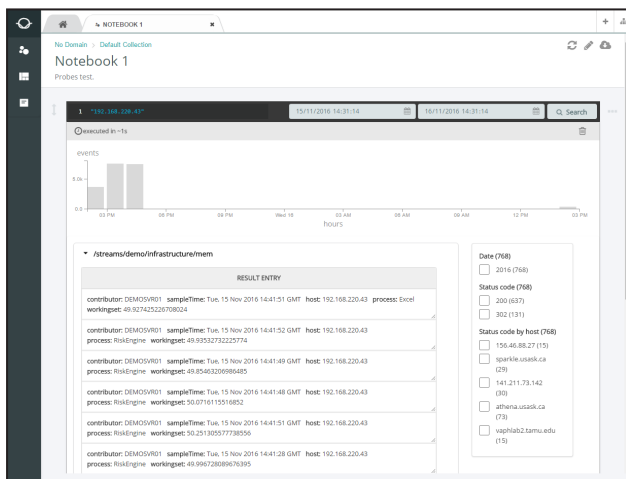


## Explore FREE TEXT Search Results

Free text search results are displayed in the results window.

If there is more than one stream in which the search string is found, these results will be presented in an expanding text box (one for each stream).

For example, searching for "192.168.220.43" in a collection of streams from Geneos returns:



A taxonomy of the results is displayed on the right. Click the check boxes to select a subset of the results and highlight them in the histogram.




## TRANSFORM DATA

Insights stores stream data immutably - the data from the original stream cannot be changed.


However, you can easily create a derived stream, even if the original data is live and not being stored as a stream. When you create a derived stream, a schema for it is created implicitly.

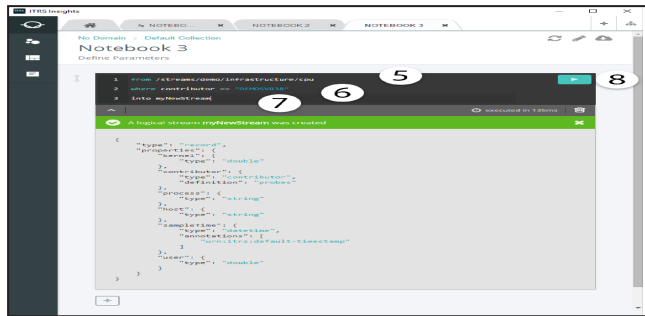
Note: a derived stream will exist only in the Notebook you created it in and will not appear in your Streams listing on the Insights Home screen. A derived stream will be lost when you exit Insights session.

### How to Create a Derived Stream

1. (Optional) Hover your cursor over the **Stream** of interest, then click **Copy to Clipboard** . This will commit a basic `from` query of that stream to the clipboard
2. On the main navigation panel, click **Notebook** 
3. From the tree, select the **Notebook** to create the derived stream within
4. Click **Add Query/Search** , then select **Query**
5. Press CTRL V on your keyboard to paste the `from` query into the **query field**, or **manually enter** the stream you wish to create the derived stream from
6. Use additional "Query Clauses" on page 46 and "Query Functions" on page 47 to specify which data to transform into the derived stream
7. Complete the query with `into`, followed by the **name** you want to give the derived stream. For example:

```
from /streams/demo/infrastructure/cpu
where contributor == "DEMOSVR38"
into myNewStream
```

8. Click **Run** . A panel will appear at the bottom of your query to confirm that the derived stream has been created



**This page intentionally blank.**

# CHAPTER 2

## ORGANIZING DATA IN ITRS INSIGHTS

## *INTRODUCTION*

To help you get the most from Insights, it's important to understand a little about the organizational concepts that underpin it, because we've developed them to make it easier for you to manage and analyse your data.

Depending on what kind of user you are - developer, front-end, systems administrator - you'll take advantage Insights' organizational constructs in different ways. A front-end user will be less concerned with how the data streams are organized and more interested in how to get to the data streams (or data sources) that matter. Conversely, a systems administrator will be more likely to focus on how the different data streams are being categorized and organized, rather than how they can be queried or searched.

Whatever your focus, however, it's worth you reading this overview from start to finish. It will only take a few minutes and it'll cement your understanding of what Insights does and why.



---

This section is intended to familiarize you with the concepts behind our product. To learn how to actually create and use the things discussed here, check out some of our user guides or our [HowTo](#) section:

[ITRS Insights Notebook, Collection & Dashboard User Guide](#)

[ITRS Insights Domain & Contributors User Guide](#)

["ITRS Insights Query, Search and Transform Introduction" on page 6](#)

[ITRS Insights visualising Results User Guide](#)

[ITRS Insights Template User Guide](#)

## *CITIES ARE GETTING SMARTER*

For the purposes of illustrating how we organize data in Insights, we're going to imagine a smart city...

A key part of smart cities will be establishing the right infrastructure to collect and intelligently analyse streams of data from connected 'things'. This is where Insights and Valo come in.

To enable you to make sense of your data and to really leverage Insights' ability to query it, we've implemented a straightforward hierarchy and organizational structure for it:

**Stream > Contributor > Domain > Tenant > Collection**

Right, let's look at how each of these works in action.

### *STREAM*

**Stream > Contributor > Domain > Tenant > Collection**

Before you can query or search your data, you have to actually get it into Insights. You do this through data **streams**.

**Stream:** append-only data (messages or events) coming from one or more external **Contributors**



### Example Stream

Arcadia Avenue a street of 20 houses all with the latest smart electricity meters, all sending information to their energy supplier HighCostEnergy.com (HCE) through an underground cable.

In the village center, HCE has a hub to transmit and process this information. Our street has a single cable running down it to the hub, but within that cable are thousands of fiber optic cables all taking information from the energy consuming devices in the house (refrigerator, toaster, TV, coffee machine, boiler, etc.).

In this example, the cable contains the stream of data that is passing to the hub, but that stream is made up of all the data coming from the individual fibre optic cables, thus **the stream of data is made up of the data from all the individual contributor devices**: the toaster at house Number 2, the refrigerator at Number 6, the coffee machine at Number 13, and so on.

One challenge when you have a number of different contributors is to

---

ensure that everyone is speaking the same language and meaning the same thing, otherwise comparisons and analytics will have limited value. To do this, we use a document of understanding (a **schema**) that defines what data the stream contains in what format. For instance, we may want the stream to provide information as data points (also known as payloads) about the time of day, the type of appliance, and the current consumption.

Because the schema ensures that the data flowing in is in a common format, we can then meaningfully compare and analyse it based on those common attribute values. We can analyse which refrigerator consumes the most energy and because our schema clearly detailed to format that the time would come in. We can compare all the appliances that are on at 8am in each house without fear that some payloads included with a time stamp of 8 would in fact be for 8pm. We can analyse pretty much anything we want, in fact.

Now for the vaguely technical bit:

Each stream has a unique name (a Unique Resource Identifier - URI) that allows us to easily categorize and identify it. In Insights, therefore, our smart house stream might be called something like

```
/streams/hce/electricity/usage
```

This URI has the form:

```
/streams/<tenant>/<collection>/<streamName>
```

Where:

`tenant` = the organizational unit the stream belongs to (see: "[Tenant](#)" on [page 39](#))

`collection` = the collection the stream is contained in

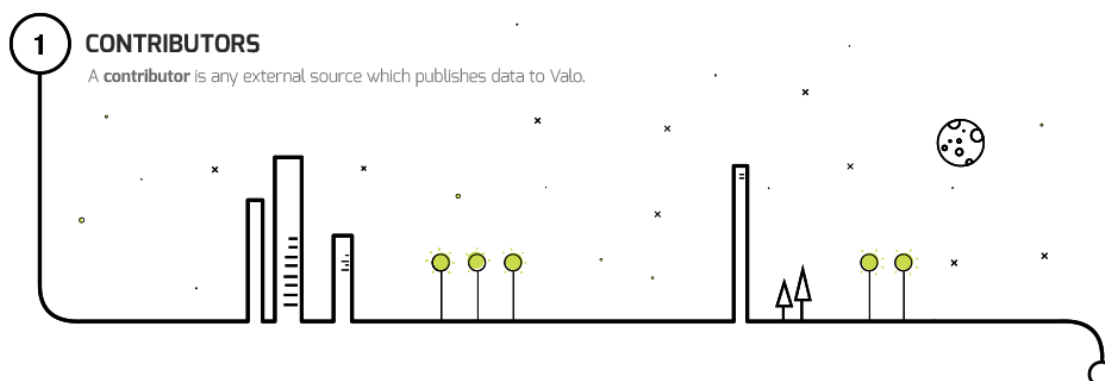
`streamName` = the name of the stream

## CONTRIBUTOR

### Stream > Contributor > Domain > Tenant > Collection

With our streams added, we can define who the external sources are. We do so with **contributors**.

**Contributor:** anything that generates and publishes data to Valo or Insights. A probe or an app are example contributors. Contributors of the same type have common characteristics but can also have different metadata to one another. For example, a probe contributor on a Windows server might be able to provide more metadata than probe contributor on a Linux server. Contributors can be grouped together by common attributes to create **Domains**.



We use Contributors to assign metadata to each data source. Assigning a data Stream as a Contributor allows us to record its static metadata (say, the make of a sensor, or the location of a probe) and then run queries against it.

For example, if the source is a sensor in a heating system, we might categorize it based on what type of heater it's in, who owns it, who it was manufactured by, or who installed it. You get the idea.

---

Like a data Stream, each Contributor has a unique URI.

```
/contributors/<tenant>/<type>
```

Where:

`tenant` = the organizational unit the stream belongs to

`type` = the type of contributor (for example probe or app)

### Example Contributor

If we imagine that the electric kettle in the kitchen at house Number 4 contained a probe, it could have a URI along the lines of  
`/contributors/hce/Num4/kettle`, for example.

### DOMAIN

#### **Stream > Contributor > Domain > Tenant > Collection**

Now that we've got our Stream(s) and Contributor(s) categorized, we can further define our data sources.

**Domain:** two or more **Contributors** grouped together dynamically by a common attribute. Domains help to organize and group different data sources in intuitive ways.

Domains are dynamic groups of Contributors.

Because Domains are dynamic, they take the attribute (metadata) of a Contributor (as opposed to a flag), and will automatically detect any new Contributors have that attribute (while ignoring those that don't).

In a smart city, we can create a Domain for all Contributors (sensors and other smart devices) in the West, and one for Contributors in the East. Any queries made to various Streams within the West domain will only operate on data coming from contributors in the West.

In an ITOA example, you might have a Domain for Windows servers within a particular IP range, and query for only those that have the tags `critical` and `error`. In this way you can run simple queries on this Domain, knowing that the appropriate data (i.e. `critical` and `error`) is always dynamically included in real-time, and that as new Windows servers are added, they'll be automatically added to the Domain.

Domains are a useful tool for organizing streaming data, especially if you consider that Contributors can publish data to multiple Streams.

So, let's group our Contributors together to form **Domains**.

### Example Domain

In our street of smart metered houses, **we could define house Number 6 as a domain, thus every smart device that is identified as being in house number 6 is considered part of that Domain for the purposes of our analysis.**

But a Domain doesn't have to be a locational entity. We could define a Domain of TVs on Arcadia Avenue, thus every contributing device that is identified as a TV that is within our street is considered part of our domain for analysis.

A powerful attribute of Insights' domains is that because the Domain is defined by an attribute of the Contributor rather than a flag that has to be set manually at installation, Domains can be dynamic, automatically picking up new contributors coming into it.

So for instance if Mr Smith in No2 Arcadia Avenue goes out and buys a new TV for his teenage son to have in his bedroom then once the TV is plugged into the system it is automatically included in the Arcadia Ave TV domain.

---

Having domains also means you can have higher level metrics or data that relate to that domain which can be published, the value of this is you can publish this higher level information to people and systems who are interested in the domain but for instance do not have permission to see all the details of the data within the domain. Again looking at our example Mr Smith could see the energy consumption of all the TVs in his house, but without having access to the energy consumption (and thus potentially the hours of watching!) that came from the new TV in his son's bedroom. Or perhaps HCE could have access to the energy consumption of each house within the street without being allowed access to the consumption of the individual devices in each house, which might be considered somewhat invasive.

A domain is used at query time to filter a stream to only include data entries for contributors and tags defined by the domain. The filter is dynamic, so a contributor which no longer matches the query will be automatically removed from the domain.

A domain definition is identified by a unique URI in the following format

```
/domain/tenant/collection/name
```

For more about Contributors and Domains, including how to create them, see [ITRS Insights Domain & Contributors User Guide](#).

*TENANT*

**Stream > Contributor > Domain > Tenant > Collection**

Still with us? Great. Only two more steps to go.

Before we can complete the organization of the data come from our smart city into Insights, we're going to define the highest level of organizational attribute: **Tenants**.

**Tenant:** the highest level of data grouping in Insights and Valo. Within a company, it may be different departments such as Sales or Engineering. If a government is collecting data from its cities, the tenant could be the city name. Tenants can be further subcategorized as **Collections**.

## Tenants Example

**A tenant is the highest level of grouping. It helps us to understand who owns the data and, if we so wish, to place virtual walls around the data limiting visibility and access.**

In our smart house example, we could say that **HCE is a tenant as it is the organization to which the stream of electricity consumption data goes:** the stream name is:

```
/streams/hce/electricity/usage.
```

100 miles away in another area of the country, there maybe another street of smart houses called Privet Drive, this road is supplied by the local company Really Expensive Electricity (REE). In this instance we may call the stream:

```
/streams/ree/electricity/usage, where REE is the tenant.
```

You can assign tenants within a company, perhaps to different departments such as Sales or Engineering. For big businesses, you may define tenants by the city or office they're located in. The bottom line is that you can organize your data how you wish.

To create a Tenant, check out our How To page: [Tenant](#).

## COLLECTION

**Stream > Contributor > Domain > Tenant > Collection**



---

Finally, we can further logically group Streams inside a tenant using **Collections**.

**Collection:** an organizational tool for grouping together two or more Streams, Domains, Dashboards or Notebooks.

### Example Collection

In our smart metered houses example, a Collection might consist of two Streams: one from Arcadia Avenue, and the other from Privet Drive.

**This page intentionally blank.**

# GLOSSARY

## C

---

### Collector

New term for a lightweight monitoring agent, which is deployed on every managed Node. See Netprobe.

### Contributor

An external source that provides one or more data streams to Valo. Contributors can be grouped together by common attributes to create Domains.

## D

---

### Dashboard

Dashboards provide an aggregated view of Data Visualizations

### Domain

Two or more Contributors grouped together by a common attribute. Domains help to organise and group different data sources in intuitive ways.

## G

---

### Gateway

A network node that provides access to another network that uses different protocols and enables transmitted data to use its routing paths.

## I

---

### Insights

A streaming big data analytics platform that simplifies the complexity involved in analysing vast amounts of data at speed. Insights combines big data storage with a real time computation engine and in-built machine learning and algorithms.

### ITRS Geneos

A real-time monitoring tool for managing increasingly complex and interconnected IT estates. Built for financial services and trading organisations, it

collects a multitude of data relating to the performance of the servers, infrastructure, connectivity and applications, analyses it to detect anything untoward, and presents it in relevant, intuitive visualisations to help diagnose and fix issues quickly.

## N

---

### Netprobe

Lightweight monitoring agent, which is deployed on every managed Node

### Notebook

A place in which to store a query or search, or a pipeline of queries or searches.

## S

---

### Schema

Document of understanding that defines what data a stream contains - its structure and shape - and in what format.

### SSR

The semi-structured repository (SSR) is based on Lucene which provides very powerful text search capabilities. Even though Lucene is geared towards indexing text, it also has very good index support for numerical data. However, if the data contains purely numerical fields the TSR might be a better fit as a repository for this kind of data.

### Stream

A stream is made up of data (messages or events) coming from one or more external contributors. Each stream has a schema that defines what information, in what format is expected. Streams are append only.

## T

---

### Tenant

The highest level of data grouping. Within a company, it may be different departments such as Sales or Engineering. If a government is collecting data from its cities, the tenant could be the city name.

### Transformational Pipeline

Each query instruction takes an input and produces an output via some form of transformation.

### TSR

The Time Series Repository (TSR) is a custom built data repository designed to handle numerical time series oriented data streams. By time series we mean a series of data points each of which has an associated time stamp.

## V

---

### Valo

Real-time analytics on data streams

# APPENDIX

## QUERY CLAUSES

### Insights Query Clauses (Transformations)

Clause / Transformation	Use	Example
from	Specifies the stream that will be the source of data for the query.	<code>from /streams/demo/geneos/cpu</code>
Where	Filters data based on a condition which follows.	<code>from /streams/demo/geneos/cpu WHERE user &gt; 50</code>
Select	Selects which fields should appear in the results.	<code>from /streams/demo/geneos/cpu SELECT host, user, kernel</code>
Aggregation	Perform aggregate calculations on the data.	<code>from historical /streams/demo/geneos/cpu SELECT avg(user) AS user, avg(kernel) AS kernel</code>
Take	Limits the Return only a certain number of results.	<code>from /streams/demo/geneos/cpu TAKE 100</code>
Order By	Orders results in ascending or descending order.	<code>from historical /streams/demo/geneos/cpu order by host ascending</code>
Group By	Group and aggregate results into distinct buckets.	<code>from /streams/demo/geneos/cpu GROUP BY host SELECT host, avg (user) AS user</code>
Time Window	Group and aggregate into windows based on a time field in the data.	<code>from /streams/demo/geneos/cpu WINDOW OF 3 seconds SELECT start, end, avg(user) as user</code>
As	Applies an alias to an output field	<code>from /streams/demo/geneos/cpu WHERE user &gt; 50 AS cpu_data</code>
Into	Gives the query a name so it can be referenced from other queries.	<code>from /streams/demo/geneos/cpu WHERE user &gt; 50 INTO result</code>
Unique	Prevents duplicate rows from appearing in the results.	<code>from /streams/demo/geneos/cpu SELECT UNIQUE host</code>
Join	Joins two streams, including over a time window.	<code>from /streams/demo/geneos/cpu WHERE user &gt; 50 INTO c INNER JOIN r ON c.host == r.host</code>
Comment	Allows descriptive text to be added to a query.	<code>from /streams/demo/geneos/cpu TAKE 10 -- This is a comment!</code>

## QUERY FUNCTIONS

ITRS Insights provides you with a significant number of query functions out of the box. These functions can be used within a query to perform complex mathematical and analytical operations on your data. See "ITRS Insights Query, Search and Transform Introduction" on page 6.

### Insights Query Functions

Function (Input)	Output	Description
"abs: absolute value" on page 55	Various	Returns the absolute value
acos(double)	double	arccosine
Anomaly	Anomaly	Identification of events which do not conform to the pattern set by other events in a dataset.
asin(double)	double	arcsine
atan(double)	double	arctangent
atan2(double, double)   double	double	arctangent with 2 arguments
avg(double)	double	average
Bivar	Bivar	Bivariate correlative statistics for use with 2 variables.
boundingBox(geoCircle)	geoRectangle	Returns geo rectangle that contains the original geo Circle
cbrt(double)	double	cubed root
ceil(double)	double	Ceiling function, returns the smallest natural number that is greater than or equal to the input
center(geoRectangle)	geoPoint	returns centre point of the geoRectangle
centerLatitude(geoCircle)	double	Gets the latitude of the centre of the circle in degrees
centerLongitude(geoCircle)	double	Gets the longitude of the centre of the circle in degrees
concat(string, string)	string	concatenation of 2 strings
"contains" on page 55	boolean	returns true if the first input includes the second (follow link for more detail)

Function (Input)	Output	Description
contributor(string)	contributor	Constructs a contributor object from a string
cos(double)	double	cosine
cosh(double)	double	hyperbolic cosine
count()	long	Increments a counter each time the function is called
countDistinct(string)	int	Counts the number of unique values
"counter" on page 56	(value:string,count:double)	Counts the number of occurrences of unique values, like collections.
crossesDateLine(geoRectangle)	boolean	returns true if area in rectangle crosses international date line
date(string)	date	converts a string into date format
datetime(string)	date	converts a string into schema format. See: <a href="#">Schema Types</a>
day(date or datetime)	integer	Returns day of the month from 1 to 31
dayOfWeek(date or datetime)	string	Returns day of the week
dayOfWeekValue(date or datetime)	integer	Returns day of the week as an integer where Monday is 1
dayOfYear(date or datetime)	int	Returns the day of the year since 1st January
days(duration)	long	Converts a duration into the number of days
diff(date or datetime , date or datetime)	duration	Difference between 2 dates/times as a duration
distance(geoPoint, geoPoint)	double	Distance between two geographical points in km, using Haversine and earth mean radius = 6371.0087714 Km
distanceDEG(geoPoint, geoPoint)	double	distance between 2 points in degrees
domain (email)	string	returns the domain name from an email address
duration(int, int, int, double)	duration	Calculates the duration based on the input arguments (days, hours, minutes, seconds)
duration(string)	duration	converts duration in string format into duration type
"EMA" on page 56	double	exponential moving average
email(string)	email	converts string into an email
erf(double)	double	error function (aka Gauss error function)
erfc(double)	double	complementary error function



Function (Input)	Output	Description
exp(double)	double	natural exponential function = $e^x$ where $e$ is Euler's number
expml(double)	double	natural exponential function minus 1 = $e^x - 1$ where $e$ is Euler's number
"first" on page 56	various	Returns the first of an ordered set (see link for more detail)
floor(double)	double	Floor function, returns the greatest natural number that is less than or equal to $x$
format(datetime, string)	string	
"geoCircle" on page 58	geoCircle	Creates a Geo circle the coordinates of its center and another point (click link for more detail)
"geoPoint" on page 58	geoPoint	Creates a geoPoint from string or longitude and latitude points
"geoRectangle" on page 58	geoRectangle	Creates a geoRectangle from various parameters
getAuthority(uri)	string	Returns the decoded authority component of this URI
getFragment(uri)	string	Returns the decoded fragment component of this URI
getHost(uri)	string	Returns the host component of this URI
getPath(uri)	string	Returns the decoded path component of this URI
getPort(uri)	integer	Returns the port number of this URI
getQuery(uri)	string	Returns the decoded query component of this URI
getScheme(uri)	string	Returns the scheme component of this URI
getUserInfo(uri)	string	Returns the user-information component of this URI
height(geoRectangle)	double	Gets the height of the rectangle in degrees latitude
Histogram		
hour(datetime or time)	integer	Hour of the day
hours(duration)	long	Number of hours
hypot(double, double)	double	Returns hypotenuse; square root of the sum of the squares of the 2 inputs
l(boolean)	integer	The indicator function. Returns 1 if true, otherwise 0
	double	Iterated (Recursive) EMA Operator
ifNull(-type-, -type-)	-type-	if first input is null use the second value

Function (Input)	Output	Description
iMicroTickFrequency	double	Average tick arrival over the given period
iMicroVolatility	double	Microscopic Volatility is computed as the norm of the microscopic derivative of the signal
iMovAverage	double	Moving Average for irregular time series
iMovDerivative	double	Moving Derivative for irregular time series
iMovDifferential	double	Moving Differential for irregular time series
iMovKurtosis	double	Moving Kurtosis for irregular time series
iMovNorm	double	Moving Norm for irregular time series
iMovOscillator	double	Classical oscillator that computes the difference between the current value and a simple EMA operator defined for a given period.
iMovSkewness	double	Returns the amount of skewness for an irregular time series
iMovStandardDeviation	double	Moving Standard Deviation for irregular time series
iMovVariance	double	Moving Variance for irregular time series
iMovOscillator	double	Moving Oscillator over an irregular time series
iMovZ	double	Moving Standardized time series (average 0, var 1)
inetAddress(string)	ip	Converts a string into an inet address
inetAddressByName(string)	ip	Creates an inet address based on a host name
isAbsolute(uri)	boolean	Returns true is the URI is absolute
isAfter(date, date) or (datetime, datetime)	boolean	Returns true if first time point is after the second
isBefore(date, date) or (datetime, datetime)	boolean	Returns true if first time point is before second
isGlobalMulticastAddress(ip)	boolean	Returns true if the multicast address has global scope
isInfinity(double)	boolean	Checks if the double value is Infinity
isIPv4(ip)	boolean	Returns true if the address is an IPv4
isIPv4Mapped(ip)	boolean	Returns true if the address is an IPv4 mapped as IPv6
isIPv6(ip)	boolean	Returns true if the address is an IPv6

Function (Input)	Output	Description
isLeapYear(date, or datetime)	boolean	Returns true if timepoint is within a leap year
isLinkLocalMulticastAddress(ip)	boolean	Returns true if the multicast address has link scope
isLoopbackAddress(ip)	boolean	Returns true if the address is an IPv4 or IPv6 loopback address
isMulticastAddress(ip)	boolean	Returns true if the address is multicast
isNan(double)	boolean	Checks if the double value is NaN (Not a Number)
isNegInfinity(double)	boolean	Checks if the double value is -Infinity
isNodeLocalMulticastAddress(ip)	boolean	Returns true if the multicast address has node scope
isNull(-type-)	boolean	returns true if value is null
isOpaque(uri)	boolean	returns true if uri is opaque
isOrganisationalMulticastAddress(ip)	boolean	Returns true if the multicast address has organization scope
isPosInfinity(double)	boolean	Checks if the double value is +Infinity
isSiteMulticastAddress(ip)	boolean	Returns true if the multicast address has site scope
kind(ip)	string	Returns a string with the type of address. "IPv4" or "IPv6"
"Last" on page 59	-type-	Returns the last in an ordered set (see link for more detail)
latitude(geoPoint)	double	returns the latitude value of a given point in degrees
lengthOfMonth(date or datetime)	integer	Returns the number of days in the month of given date
lengthOfYear(date or datetime)	integer	Returns the number of days in the year of given date (365 or 366)
log(double)	double	Returns the natural logarithm (base e) of the input
log10(double)	double	returns the common logarithm (base 10) of input
log1p(double)	double	Returns the natural logarithm of the the argument plus 1
longitude(geoPoint)	double	returns the longitude value of a given point in degrees
"max" on page 60	integer	Returns the largest argument for each payload
"Max" on page 62	-type-	Aggregates payloads into a single result
micros(duration)	long	Converts duration into microseconds
millis(duration)	long	Converts duration into milliseconds

Function (Input)	Output	Description
"min" on page 63	integer	Returns the smallest argument for each payload
"Min" on page 64	-type-	Aggregates payloads into a single result
minus(date, duration) or (datetime, duration)	datetime	returns the date and time of the input date minus the duration
minus(time, duration)	time	returns the time of the input time minus the duration
minute(time) or (datetime)	integer	returns the minutes digits in the time specified
minuteofDay(time) or (datetime)	integer	minute in the day of the specified time
minutes(duration)	long	duration converted into minutes
month(time) or (datetime)	integer	month of the year as an integer where Jan =1, Feb = 2,....
multicastKind(ip)	string	Returns a string with the type of multicast address [Global, Link, Node, Organisation, Site or Unknown]
nameOfMonth(date) or (datetime)	string	Returns the name of the month, as a string
nano(datetime) or (time)	integer	Gets the fractional second in nano units
nanos(duration)	long	converting a duration into nanoseconds
ordinal(int, int, int, int)	date	Calculates a date based on a ordinal position where arguments are (position, day of week, month, year)
ordinal(int, int, date)	date	Calculates a date based on a ordinal position and within the month and year given by the last parameter where arguments are (position, day of the week, date in the month and year)
plus(date, duration) or (datetime, duration)	datetime	returns what the datetime would be after the duration if starting at the input time
plus(time, duration)	time	returns what the time would be after the duration if starting at the input time
pointOnBearing(geoPoint, double, double)	geoPoint	Returns location as reported at a distance (2nd argument) and bearing (3rd argument) from a known point (1st argument)
pow(double, double)	double	returns the value of the first input to the power of the second
"prefix" on page 65	boolean	Returns true if the 1st string is prefixed by the specified (2nd) character sequence
Quantiles	sequence	
radiusDEG (geoCircle)	double	radius of the geo circle in degrees
radiusKM (geoCircle)	double	radius of the geo circle in kilometres

Function (Input)	Output	Description
relate(geoPoint, geoCircle) or (geoPoint, geoRectangle)	String{"Contains" or "Disjoint" or "Intersects" or "Within"}	returns 1 of 4 options of how the point relates to the geo shape
search(string)	boolean	Support for fuzzy search over all string fields in a payload
second(datetime) or (time)	int	returns the second field from the time
secondOfDay(datetime) or (time)	int	returns the number of seconds since the previous midnight
seconds(duration)	long	converts duration into seconds
sin(double)	double	sine
sinh(double)	double	hyperbolic sine function
sqrt(double)	double	Square root
startOfHour(datetime) or (time)	datetime or time	Strips minute, second and sub-second information from the date time
startOfMinute(datetime) or (time)	datetime or time	Strips second and sub-second information from the date time
startOfSecond(datetime) or (time)	datetime or time	Strips sub-second information from the date time
"sum" on page 66	long	
sum	double	
tan(double)	double	tangent
tanh(double)	double	hyperbolic tangent
time( <i>string</i> )	time	Constructs a time object from a string
toBool(-type-)	boolean	converts integer, double, long, short, byte or string to boolean
toByte(-type-)	byte	converts integer, double, long, short, boolean or string to byte
toDate(datetime)	date	converts datetime to date
toDateTime(date)	datetime	converts date to datetime, making the time the earliest possible (e.g 0:00:00...)
toDegrees(double)	double	converts radians to degrees
toDouble(-type-)	double	converts a long, integer, short, byte, boolean or string to a double
toInt(-type-)	integer	converts a long, double, short, byte, boolean or string to an integer
toLong(-type-)	long	converts a double, integer, short, byte, boolean or string to a long

Function (Input)	Output	Description
toLower(string)	string	converts string to lower case
toRadians(double)	double	converts degrees to radians
toShort(-type-)	short	converts a double, integer, long, byte, boolean or string to a short
toString(uuid)	string	Converts a UUID to String
toTime(datetime)	time	converts datetime into time
toUpper(string)	string	converts string to uppercase
<b>Univar</b>		
uri(string)	uri	converts string to uri
utcDate()	date	Returns the current date at UTC
utcNow()	datetime	returns current date and time at UTC
utcTomorrow()	date	returns the date one day after the current date at UTC
utcYesterday()	date	returns the date one day before the current date at UTC
uuid(string)	uuid	converts a string to a uudi
variant(uuid)	integer	The variant number describing the layout of the UUID
version(uuid)	integer	The version number describing how this UUID was generated
weekOfMonth(date) or (datetime)	integer	returns the week of the month
weekOfYear(date) or (datetime)	integer	returns the week of the year
width(geoRectangle)	double	Gets the width of the rectangle in degrees longitude
year(date) or (datetime)	integer	returns the year of the date input

*ABS: ABSOLUTE VALUE*

Returns the absolute value of the input

Input	Output
double	double
long	long
integer	integer
short	integer
byte	integer

*CONTAINS*

Input	Output	Explanation
(string, string)	boolean	True if the first string contains the character sequence of the second
(geoPoint, geoCircle)	boolean	True if the geo circle contains the geoPoint
(geoPoint, geoRectangle)	boolean	True if the geo rectangle contains the geoPoint

## COUNTER

Counts the number of occurrences of unique values. Returns all the values seen along with their counts (a histogram essentially).

Format: counter()(value:string,count:long)

## EMA

Exponential Moving Average with next point interpolation

Format: EMA(period)(timestamp, value)

## PARAMETERS

Name	Type	Comments
period	double	Range of the operator in units of time
timestamp	datetime	Timestamp associated with the point being processed.
value	double	value of the time series at the given timestamp

## FIRST

Returns the first in an ordered set.



FORMAT

Formats datetime into a string

FORM:

```
format(_datetime_, format)
```

Where format is a string of one of the following

Supported Formats
BASIC_ISO_DATE
ISO_DATE
ISO_DATE_TIME
ISO_INSTANT
ISO_LOCAL_DATE
ISO_LOCAL_DATE_TIME
ISO_LOCAL_TIME
ISO_OFFSET_DATE
ISO_OFFSET_DATE_TIME
ISO_OFFSET_TIME
ISO_ORDINAL_DATE
ISO_TIME
ISO_WEEK_DATE
ISO_ZONED_DATE_TIME

## GEOCIRCLE

Creates a Geo circle the coordinates of its center and another point

Input	Output	Explanation
(geoPoint, geoPoint)	geoCircle	Creates a Geo circle the coordinates of its centre the first point and a point on it circumference set by the second
(geoPoint, double)	geoCircle	Creates a Geo circle the coordinates of its centre the first point and a radius of the second argument in km
(double, double, double)	geoCircle	Creates a Geo circle with its centre at longitude of the first argument, latitude the second argument and radius the third
(string)	geoCircle	Creates a Geo circle using a string of format [38.898648N 77.037692W 10.0]

## GEOPOINT

Input	Output	Explanation
(double, double)	geoPoint	Creates a Geo Point by longitude and latitude
(string)	geoPoint	Creates a Geo Point by passing a string like [28.898648N 77.037692W]

## GEORECTANGLE

Creates a geoRectangle from various parameters

Input	Output	Explanation
(string)	geoRectangle	Creates a Geo Rectangle by passing a string like [28.898648N 77.037692W 38.898648N 67.037692W]
(double, double, double, double)	geoRectangle	Creates a Geo Rectangle by specifying all four coordinates in the order longitude min, latitude min, longitude max and latitude max
(geoPoint, geoPoint)	geoRectangle	Creates a Geo Rectangle by specifying two opposite corners where the first argument is the lower left corner and the second the upper right

*LAST*

Returns the last in an ordered set.

MAX

Input	Output
(double, double)	double
(long, long)	long
(integer, integer)	integer
(short, short)	integer
(byte, byte)	integer

So for Data:

```
{a: 1, b: 2}, {a: 3, b: 1}, {a: 2, b: 4}
```

Realtime query

```
from ...
  select max(a, b) as r:
```

would return

```
{r: 2}, {r: 3}, {r: 4}
```

Historical query

```
from historical ...
  select max(a, b) as r:
```

would return

```
{r: 2}, {r: 3}, {r: 4}
```

```
from /streams/demo/infrastructure/cpu
select host, max(user, kernel) as max, (user + kernel) * 100 as f
```

MAX

Input	Output
(byte)	byte
(short)	short
(integer)	integer
(double)	double
(long)	long
(datetime)	datetime

So for example with the data:

```
{a: 1, b: 2}, {a: 3, b: 1}, {a: 2, b: 4}
```

Realtime query

```
from ...
  select max(a) as r
```

would return

```
{r: 1}, {r: 3}, {r: 3}
```

and a historical query

```
from historical ...
select max(b) as r
```

would return

```
{r: 4}
```

MIN

Input	Output
(double, double)	double
(long, long)	long
(integer, integer)	integer
(short, short)	integer
(byte, byte)	integer

So for Data:

```
{a: 1, b: 2}, {a: 3, b: 1}, {a: 2, b: 4}
```

Realtime query

```
from ...
  select min(a, b) as r:
```

would return

```
{r: 1}, {r: 1}, {r: 2}
```

Historical query

```
from historical ...
  select min(a, b) as r:
```

would return

`{r: 1}, {r: 1}, {r: 2}`

*MIN*

Input	Output
(byte)	byte
(short)	short
(integer)	integer
(double)	double
(long)	long
(datetime)	datetime



### *Example*

So for example with the data:

```
{a: 1, b: 2}, {a: 3, b: 1}, {a: 2, b: 4}
```

#### Realtime query

```
from ...  
  select min(a) as r
```

would return

```
{r: 1}, {r: 1}, {r: 1}
```

#### and a historical query

```
from historical ...  
select min(b) as r
```

would return

```
{r: 1}
```

### *PREFIX*

True if the string is prefixed by the specified character sequence. Format: `prefix(string: String, sequence: String)`

### *Example:*

```
from historical /streams/demo/system/probes
```

```
where prefix(os.name, "win") || prefix(os.name, "ma")
```

## *SUM*

Sum of the values

### *Example:*

```
from /streams/demo/infrastructure/cpu  
select sum(user) as totalU emit every value
```



QUERY OPERATORS

When processing query expressions, Insights uses the table below to determine the order of operations.

Insights Query Operators

Precedence	Operators	Description
1	()	Explicit precedence
2	* / %	Multiplication, division, and remainder
3	+ -	Addition and subtraction
4	< <= == != >= >	Comparison
5	NOT	Logical not
6	AND	Logical and
7	OR	Logical or

Using the table:

Operators with a lower precedence are evaluated before operators with a higher precedence.

Operators with the same precedence are evaluated in order of appearance in the query expression.