Lookup

A SEARCH SERVER

Shauryadeep Chaudhuri | May 18, 2016

Lookup Storage

Once Lookup has been installed and the minimal configuration required has been specified in the configuration file, you would be able to use Lookup right away.

The storage location of documents/datastore of Lookup is called **Warehouse.** The Warehouse path can be set in the configuration file.

Lookup has a 3 level document hierarchy, namely

Index -> Similar to Databases in a SQL Database

Schema -> Similar to Tables in a SQL Database

Entry -> Similar to Records/Tuples in a SQL Database

In Lookup the Schema structure is not enforced on all of its Entries. Rather the Schema structure is the collective structure of all its entries and is auto updated with new entries.

Start Looking Up Right Away

All the CRUD operations supported by Lookup are enabled by a ReSt API of Lookup.

JSON Queries need to be sent using these methods via a ReSt Client, in addition to you can also retrieve entries via URL in a browser. More on that later.

In the examples we would assume that Lookup has been set on **Port 7123** on localhost. And would post using Curl commands. You can use any rest client.

Entries are inside of Schema, which are contained in Index. So you would need to create an index, then a schema, then Entries.

Lookup identifies queries as JSON, so a proper json needs to be sent to the API.

There are different Identfiers for each type, for lookup to understand the query

index -> Index -> Index name which would be created, fetched or deleted.

schema -> Schema -> Schema which would be created, fetched or deleted.

entry -> Entry -> Entry which would be created, fetched, deleted or updated.

query -> Query -> Query which would be used to fetch ,update or delete Entries.

Any other field -> Entry Fields -> If any other field is mentioned then it would result in an error except when creating Entries. Additional fields will be identified as fields which are to be inserted into Entries.

Create – PUT method – on root address

Documents are identified using their names, so two documents cannot be created with the same name.

Index

To create an Index named “MyFirstIndex”, the following query needs to be sent to the API.

```
curl -XPUT "http://localhost:7132/" –d '

{"index":"MyFirstIndex"} '
```

If the creation is successful it will return a message saying it has been added successfully, else it would show the error.

Schema

To create a Schema named “MyFirstSchema”, the following query needs to be sent to the API.

```
curl -XPUT "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”: MyFirstSchema } '
```

If the creation is successful it will return a message saying it has been added successfully, else it would show the error.

Entry

To create an Entry named “Entry1”, the following query needs to be sent to the API. Supported Field Types for Entries are -: **String, Integer, Float, Long and Boolean**

```
curl -XPUT "http://localhost:7132/" –d '

   {"index":"MyFirstIndex",”schema”: MyFirstSchema , “entry”:”entry1”,”field1”:22,”field2”:”Some Random String”,”field3”:9.23} '
```

If the creation is successful it will return a message saying it has been added successfully, else it would show the error.

Fetch – GET method – on root address

Using Document Name

Documents can be read by explicitly providing their names, Entries can also be searched by providing a query.

Index

To fetch an Index named “MyFirstIndex”, the following query needs to be sent to the API.

```
curl -XGET "http://localhost:7132/" –d '

{"index":"MyFirstIndex"} '
```

If it can find the index, it would return with index name and schemas it consists, else it would return a message saying that nothing has been found.

Schema

To fetch a Schema named “MyFirstSchema”, the following query needs to be sent to the API.

```
curl -XGET "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”: MyFirstSchema } '
```

If it can find the Schema, it would return with Schema name, Entries it consists, and the fields that are present in the Schema,else it would return a message saying that nothing has been found.

Entry

To fetch a Schema named “entry1”, the following query needs to be sent to the API.

```
curl -XGET "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”: MyFirstSchema , “entry”:”entry1”} '
```

If it can find the Entry, it would return with Entry name, the fields and their values, and its version. The version is incremented with each update of the document.

Fetch – GET method – on root address

Using Lookup Query Language

Lookup can retrieve entries based on a query specified by you. Though you would still need to specify the Index & Schema name, Lookup Query Language allows to use different operators on fields to retrieve entries which meet the specified conditions.

In the below examples we would use ExField as the field name.

Operator	Description	Usage
== **Equality Operator**	Retrieves Entries with the exact value specified for field	ExField==9 ExField==”Something”
< **Less Than**	Retrieves Entries with the values less than value specified for field	ExField<9
<= **Less Than Equal to**	Retrieves Entries with the values less than and equal to value specified for field	ExField<=9
> **Greater than**	Retrieves Entries with the values greater than value specified for field	ExField>9
>= **Greater Than Equal to**	Retrieves Entries with the values greater than or equal to value specified for field	ExField>=9

NOTE: Since Lookup has been built in Python, Comparison operators will behave as they do in Python. String comparison using operators other than equality will have behavior as mentioned in Python Docs.

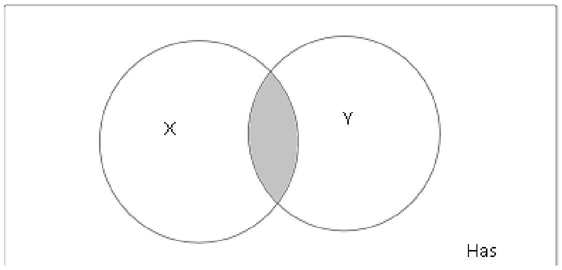
The comparison uses lexicographical ordering: first the first two items are compared, and if they differ this determines the outcome of the comparison; if they are equal, the next two items are compared, and so on, until either sequence is exhausted.

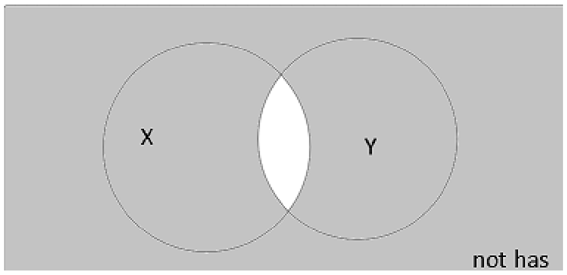
Several Keywords are also used in Lookup Query Language

Keyword	**Description**	**Usage**
has **Mostly used for String Field Searches**	This keyword will check if the field has the combination of values specified. Values are comma seperated	ExField has paul,mom
not has	This is the opposite of has. It would find the entries which do not have the combination of specified words.	ExField not has paul,mom
in	This keyword will return entries with the field matching any of specified values	ExField in paul,mom
not in	This keyword will return entries with the field not matching any of specified values	ExField not in paul,mom
AND **Logical AND**	This Keyword will allow the conjunction of two conditions. Entries need to have both condition satisfied	ExField1>8 AND ExField2 has paul,mom
OR **Logical OR**	This Keyword will allow the disjunction of two conditions. Entries need to have both condition satisfied	ExField1>8 OR ExField2 has paul,mom
() **Parenthesis**	Parenthesis will allow to make complex queries and allow to evaluate a sub condition before evaluating the whole.	(ExField1>8 OR ExField2 has paul,mom) AND ExField3==99

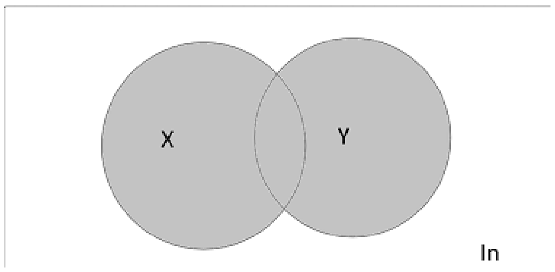
The Has & the In Keyword can be confusing with textual description before using it. Both the keywords have been explained using venn diagrams below. Assuming two field values x and y were passed.

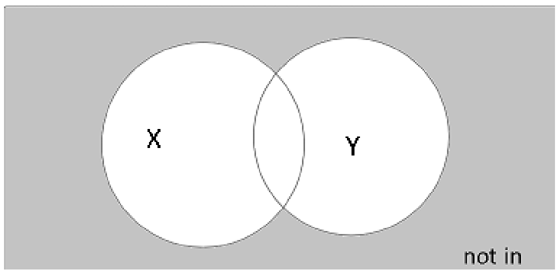
Has & Not Has





In & Not In





Example Lookup Query using CURL

```
curl -XGET "http://localhost:7132/" –d '
  {"index":"MyFirstIndex",”schema”: MyFirstSchema , “query”:”(ExField1>8 OR ExField2 has paul,mom) AND ExField3==99”} '
```

GET method also works via browser URL in the following way.

```
http://localhost:7123/index/schema/entry
```

```
http://localhost:7123/index/schema/query=LQL
```

Example:

```
http://localhost:7123/MyFirstIndex/MyFirstSchema/entry1
```

```
http://localhost:7123/MyFirstIndex/MyFirstSchema/query= ExField1>8 OR ExField2 has paul,mom) AND ExField3==99
```

Update – POST method – on root address

Only Field Values of Entries can be updated. Updating an Entry can be done by using a query similar to query used to fetch the entries, and the fields that would need to be updated. If the Entry does not have a value for the specified for the field then the field be inserted in the entry. Update can be done on multiple Entries using a single query using Lookup Query Language.

Update Query using Entry Name

```
curl -XPOST "http://localhost:7132/" –d '

  {"index":"MyFirstIndex",”schema”: MyFirstSchema , “entry”:”entry1”,”field1”:22,”field2”:”Some Random String”,”field3”:9.23} '
```

Update Query using LQL

```
curl -XPOST "http://localhost:7132/" –d '

  {"index":"MyFirstIndex",”schema”: MyFirstSchema , “query”:” (ExField1>8 OR ExField2 has paul,mom) AND ExField3==99”,”field1”:”New value”,”field2”:9.22} '
```

The Type of a field would be freezed when it has been initially created and cannot be changed after that.

```
curl -XGET "http://localhost:7132/" –d '

{"index":"MyFirstIndex"} '
```

Delete – DELETE method – on root address

The Delete method uses the exact query that has been used to fetch the results. As you may expect instead of retrieving Index, Schema or Entry, it would delete them. Lookup Query Language can also be used in this method as well.

In Addition to deleting these 3 documents, you can also erase a field from a schema. This would result in the field being deleted from all the entries that contain the field and also from the schema document.

Example Queries:

Index:

```
curl -XDELETE "http://localhost:7132/" –d '

 {"index":"MyFirstIndex"} '
```

Schema:

```
curl - XDELETE "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”: “MyFirstSchema” } '
```

Field:

```
curl - XDELETE "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”:”MyFirstSchema”,”field”:”fieldname” } '
```

Entry:

```
curl - XDELETE  "http://localhost:7132/" –d '

 {"index":"MyFirstIndex",”schema”: MyFirstSchema , “entry”:”entry1”} '
```

Using LQL

```
curl - XDELETE "http://localhost:7132/" –d '

   {"index":"MyFirstIndex",”schema”:  MyFirstSchema  ,  “query”:”(ExField1>8  OR  ExField2  has
paul,mom) AND ExField3==99”} '
```

This Brings to an end to the Lookup Tutorial

Happy Looking Up ☺