RediSearch quick start tutorial

Prerequisites

For this quick start tutorial, you need:

- A Redis database with the RediSearch module enabled. You can use either:
  - A Redis Cloud database
  - A Redis Enterprise Software database
- redis-cli command line tool
- redis-py client library v4.0.0 or greater

RediSearch with redis-cli

The redis-cli command line tool comes packaged with Redis. You can use it to connect to your Redis database and test the following RediSearch commands.

Connect to a database

```bash
$ redis-cli -h <endpoint> -p <port> -a <password>
127.0.0.1:12543>
```

Create an index

When you define an index, you also need to define the schema, or structure, of the data you want to add to the index.

The schema for this example has four fields:

- title(TEXT)
- body(TEXT)
- url(TEXT)
- visits(NUMERIC)

Use FT.CREATE to create an index and define the schema:

```bash
127.0.0.1:12543> FT.CREATE database_idx PREFIX 1 "doc:" SCORE 0.5 SCORE_FIELD "doc_score" SCHEMA title TEXT body TEXT url TEXT visits NUMERIC
```

This command indexes all hashes with the prefix "doc:". It will also index any future hashes that have this prefix.

By default, all documents have a score of 1.0. However, you can configure an index’s default document score with the SCORE <default_score> option.

If you want to allow document-specific scores, use the SCORE_FIELD <score_attribute> option.
Add documents

After you create an index, you can add documents to the index.

Use the HSET command to add a hash with the key "doc:1" and the fields:

- title: "RediSearch"
- body: "RediSearch is a powerful indexing, querying, and full-text search engine for Redis"
- url: https://redis.com/modules/redis-search/
- visits: 108

```bash
127.0.0.1:12543> HSET doc:1 title "RediSearch" body "RediSearch is a powerful indexing, querying, and full-text search engine for Redis" url "<https://redis.com/modules/redis-search/>" visits 108 (integer) 4
```

If you want to make a document appear higher or lower in the search results, add a document-specific score.

To do so, use the score_attribute set by the SCORE_FIELD option during index creation (score_attribute is doc_score in this case):

```bash
127.0.0.1:12543> HSET doc:2 title "Redis" body "Modules" url "<https://redis.com/modules>" visits 102 doc_score 0.8 (integer) 5
```

Search an index

Search the index for any documents that contain the word "primary":

```bash
127.0.0.1:12543> FT.SEARCH database_idx "search engine" LIMIT 0 10
1) (integer) 1
2) "doc:1"
3) 1) "title"
   2) "RediSearch"
   3) "body"
   4) "RediSearch is a powerful indexing, querying, and full-text search engine for Redis"
   5) "url"
   6) "<https://redis.com/modules/redis-search/>"
   7) "visits"
   8) "108"
```

Drop an index

You can remove the index without deleting any associated documents with the FT.DROPINDEX command:

```bash
127.0.0.1:12543> FT.DROPINDEX database_idx
OK
```

Auto-complete

You can use RediSearch suggestion commands to implement auto-complete.
Add a phrase for the search engine to suggest during auto-completion:

```
127.0.0.1:12543> FT.SUGADD autocomplete "primary and caching" 100
"(integer)" 1
```

Test auto-complete suggestions:

```
127.0.0.1:12543> FT.SUGGET autocomplete "pri"
1) "primary and caching"
```

**RediSearch with Python**

If you want to use RediSearch within an application, you can use one of these client libraries.

The following example uses the Redis Python client library `redis-py`, which supports RediSearch commands as of v4.0.0.

This Python code creates an index, adds documents to the index, displays search results, and then deletes the index:
import redis
from redis.commands.search.field import TextField, NumericField
from redis.commands.search.indexDefinition import IndexDefinition
from redis.commands.search.query import Query

# Connect to a database
r = redis.Redis(host="<endpoint>", port="<port>",
    password="<password>")

# Options for index creation
index_def = IndexDefinition(
    prefix = ['py_doc:'],
    score = 0.5,
    score_field = "doc_score"
)

# Schema definition
schema = (TextField("title"),
    TextField("body"),
    TextField("url"),
    NumericField("visits")
)

# Create an index and pass in the schema
r.ft("py_idx").create_index(schema, definition = index_def)

# A dictionary that represents a document
doc1 = { "title": "RediSearch",
    "body": "RediSearch is a powerful indexing, querying, and full-text search engine for Redis",
    "url": "<https://redis.com/modules/redi-search/>",
    "visits": 108
}
doc2 = { "title": "Redis",
    "body": "Modules",
    "url": "<https://redis.com/modules>",
    "visits": 102,
    "doc_score": 0.8
}

# Add documents to the database and index them
r.hset("py_doc:1", mapping = doc1)
r.hset("py_doc:2", mapping = doc2)

# Search the index for a string; paging limits the search results to 10
result = r.ft("py_idx").search(Query("search engine").paging(0, 10))

# The result has the total number of search results and a list of documents
print(result.total)
print(result.docs)

# Delete the index; set delete_documents to True to delete indexed documents as well
r.ft("py_idx").dropindex(delete_documents=False)
$ python3 quick_start.py
1
[Document {'id': 'py_doc:1', 'payload': None, 'visits': '108', 'title': 'RediSearch', 'body': 'RediSearch is a powerful indexing, querying, and full-text search engine for Redis', 'url': '<https://redis.com/modules/redis-search/>'}]

More info

- RediSearch commands
- RediSearch client libraries

Updated: March 11, 2022