Indexing / querying JSON documents

Adding a JSON document to an index

[1]:
```
import redis
from redis.commands.json.path import Path
import redis.commands.search.aggregation as aggregations
import redis.commands.search.reducers as reducers
from redis.commands.search.field import TextField, NumericField, TagField
from redis.commands.search.indexDefinition import IndexDefinition, IndexType
from redis.commands.search.query import NumericFilter, Query


r = redis.Redis(host='localhost', port=6379)
user1 = {
    "user":{
        "name": "Paul John",
        "email": "paul.john@example.com",
        "age": 42,
        "city": "London"
    }
}
user2 = {
    "user":{
        "name": "Eden Zamir",
        "email": "eden.zamir@example.com",
        "age": 29,
        "city": "Tel Aviv"
    }
}
user3 = {
    "user":{
        "name": "Paul Zamir",
        "email": "paul.zamir@example.com",
        "age": 35,
        "city": "Tel Aviv"
    }
}

user4 = {
    "user":{
        "name": "Sarah Zamir",
        "email": "sarah.zamir@example.com",
        "age": 30,
        "city": "Paris"
    }
}
r.json().set("user:1", Path.root_path(), user1)
r.json().set("user:2", Path.root_path(), user2)
r.json().set("user:3", Path.root_path(), user3)
r.json().set("user:4", Path.root_path(), user4)

schema = (TextField("$.user.name", as_name="name"),TagField("$.user.city", as_name="city"
r.ft().create_index(schema, definition=IndexDefinition(prefix=["user:"], index_type=Inde
```

[1]:
```
b'OK'
```

Searching

Simple search

[2]:
```
r.ft().search("Paul")
```

[2]:
```
Result{2 total, docs: [Document {'id': 'user:1', 'payload': None, 'json': '{"user":{"name
```

Filtering search results

[3]:
```
q1 = Query("Paul").add_filter(NumericFilter("age", 30, 40))
r.ft().search(q1)
```

```
[3]:  Result{1 total, docs: [Document {'id': 'user:3', 'payload': None, 'json': '{"user":{"name
```

Paginating and Ordering search Results

```
[4]:  # Search for all users, returning 2 users at a time and sorting by age in descending orde
      offset = 0
      num = 2
      q = Query("*").paging(offset, num).sort_by("age", asc=False) # pass asc=True to sort in a
      r.ft().search(q)
```

```
[4]:  Result{4 total, docs: [Document {'id': 'user:1', 'payload': None, 'age': '42', 'json': '{
```

Counting the total number of Items

```
[5]:  q = Query("*").paging(0, 0)
      r.ft().search(q).total
```

```
[5]:  4
```

Projecting using JSON Path expressions

```
[6]:  r.ft().search(Query("Paul").return_field("$.user.city", as_field="city")).docs
```

```
[6]:  [Document {'id': 'user:1', 'payload': None, 'city': 'London'},
       Document {'id': 'user:3', 'payload': None, 'city': 'Tel Aviv'}]
```

Aggregation

```
[7]:  req = aggregations.AggregateRequest("Paul").sort_by("@age")
      r.ft().aggregate(req).rows
```

```
[7]:  [[b'age', b'35'], [b'age', b'42']]
```

Count the total number of Items

```
[8]:  # The group_by expects a string or list of strings to group the results before applying t
      # each group. Passing an empty list here acts as `GROUPBY 0` which applies the aggregatio
      req = aggregations.AggregateRequest("*").group_by([], reducers.count().alias("total"))
      r.ft().aggregate(req).rows
```

```
[8]:  [[b'total', b'4']]
```