Couchbase: NoSQL & The Bridge Between

Nic Raboy, Developer Advocate at Couchbase

About Me

Nic Raboy

Developer Advocate at Couchbase

@nraboy (Twitter)

Node.js, Android, Java, Ionic Framework, NoSQL, SQL

There are so many databases available, so what do I choose?

NoSQL?

Polyglot persistence is "...using multiple data storage technologies, chosen based upon the way data is being used by individual applications. Why store binary images in relational database, when there are better storage systems?"

Martin Fowler and Pramod Sadalage

Key-Value

email: nic@couchbase.com

Key-Value

email: {

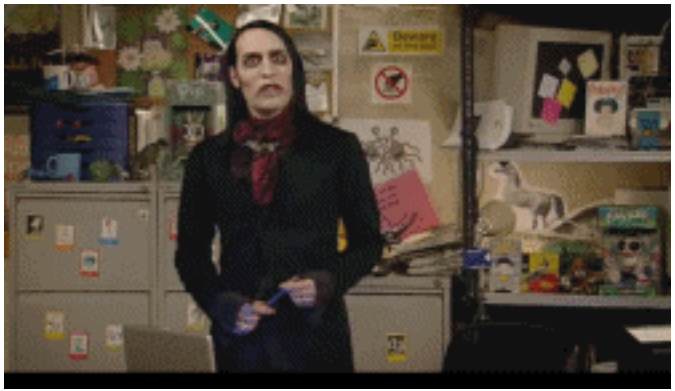
“work”: “nic@couchbase.com”,

“other”: “nic.raboy@couchbase.com”

}

Key-Value

email:



Document

nic@couchbase.com

```
{
    “city”: “San Francisco”,
    “glasses”: true,
    “team”: “Developer Advocacy”,
    “music”: “Pop!”
}
```

Document

London

matthew@couchbase.com
laura@couchbase.com
martin@couchbase.com
james@couchbase.com
tom@couchbase.com
david@couchbase.com

Developer Advocacy

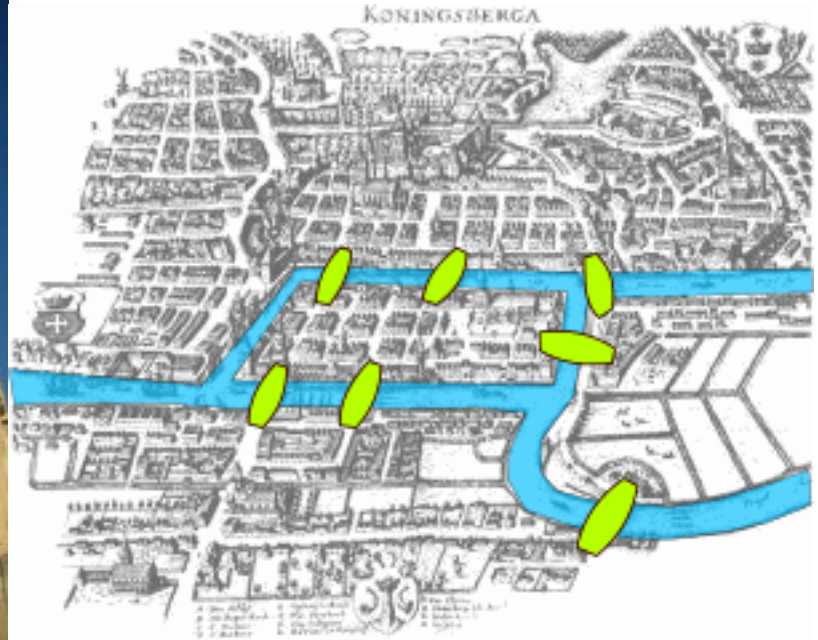
nic@couchbase.com
matt@couchbase.com
william@couchbase.com
matthew@couchbase.com
martin@couchbase.com
laurent@couchbase.com

Document

London & Developer Advocacy

matthew@couchbase.com
laura@couchbase.com
james@couchbase.com
martin@couchbase.com

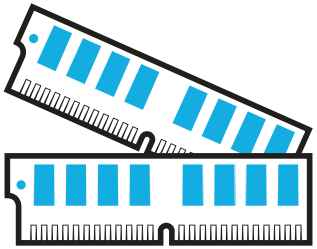
The Others: Column & Graph



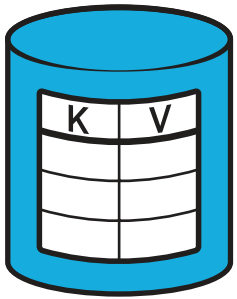
Where Does Couchbase Fit In?

Couchbase provides a complete Data Management solution

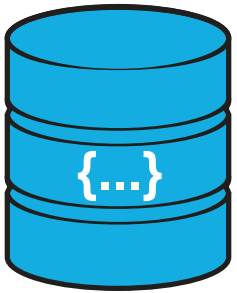
Designed and optimized for interactive web applications



High availability cache

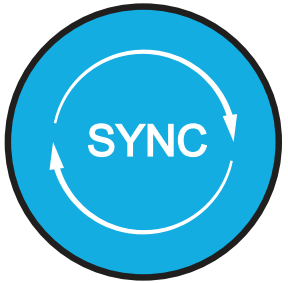


Key-value store



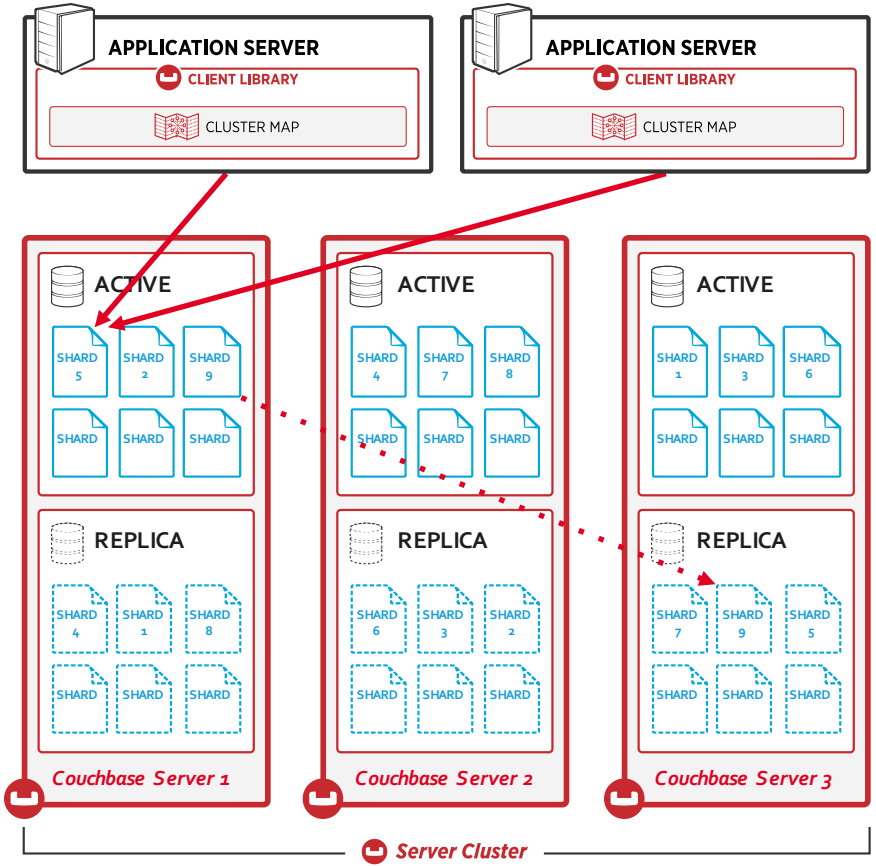
Document database

Embedded database



Sync management

Basic Operation

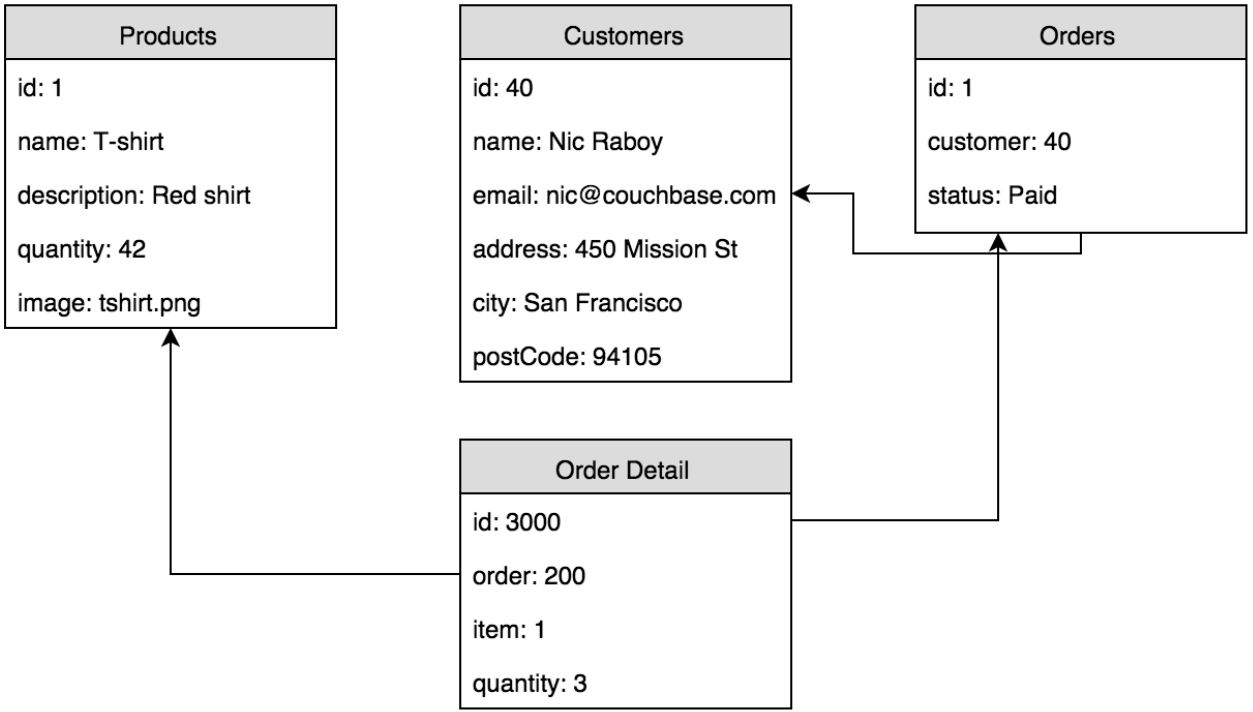


Application has single logical connection to cluster (client object)

- Data is automatically sharded resulting in even document data distribution across cluster
- Each vbucket replicated 1, 2 or 3 times ("peer-to-peer" replication)
- Docs are automatically hashed by the client to a shard
- Cluster map provides location of which server a shard is on
- Every read/write/update/delete goes to same node for a given key
- Strongly consistent data access ("read your own writes")
- A single Couchbase node can achieve 100k's ops/sec so no need to scale reads

Relational to NoSQL Document

An Ecommerce Order - RDBMS



Referred Documents

Embedded Document

When to embed

- You should embed data when:
  - Speed trumps all else
  - Slow moving data
  - Application layer can keep multiple copies of same data in sync

When to refer

- You should refer to data when:
  - Consistency is a priority
  - The data has large growth potential

Going Deeper…

Designing Documents & Their Keys

Three ways to build a key

- Key design is as important as document design.
- There are three broad types of key:
  - Human readable/deterministic: e.g. an email address
  - Computer generated/random: e.g. UUID
  - Compound: e.g. UUID with a deterministic portion

Human readable/deterministic

Key: nic@couchbase.com

```
public class user {

      private String name;
      private String email;
      private String streetAddress;
      private String city;
      private String country;
      private String postCode;
      private Array orders;
      private Array productsViewed;
}
```

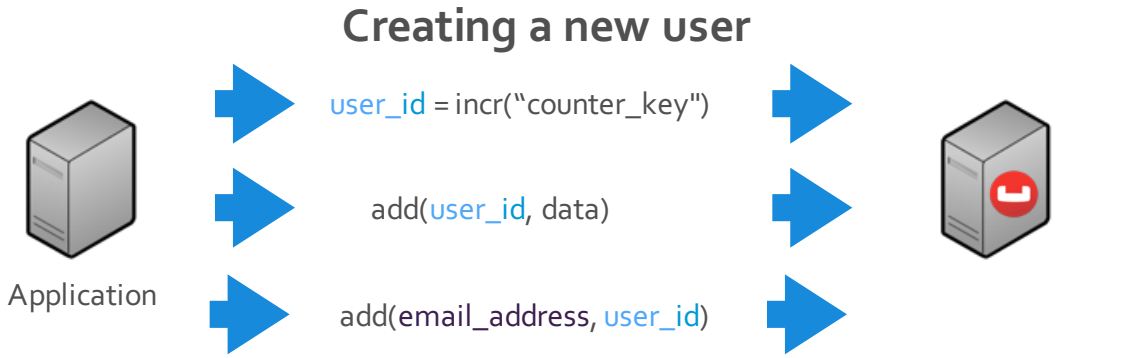
```
{
   "name": ”Nic Raboy",
   "address": ”450 Mission Street",
   "city": ”San Francisco",
   "postCode": ”94105",
   "orders": [ 1, 9, 698, 32 ],
   “productsViewed”: [8, 33, 99, 100]
 }
```

Random/computer generated

Key: 1001

```
{
   "name": ”Nic Raboy",
   "email": ”nic@couchbase.com",
   "address": ”450 Mission Street",
   "city": ”San Francisco",
   "postCode": ”94105",
   "orders": [ 1, 9, 698, 32 ],
   “productsViewed”: [8, 33, 99, 100]
 }
```

Using counters to generate keys



Multiple look-up documents

u::count

1001

u::1001

```
{ "name": “Nic Raboy",
"facebook_id": 16172910,
"email": “nic@couchbase.com”,
“password”: ab02d#Jf02K
"created_at": "5/1/2012 2:30am",
“facebook_access_token”: xox0v2dje20,
“twitter_access_token”: 20jffieieaaixixj }
```

fb::16172910

1001

nflx::2939202

1001

twtr::2920283830

1001

em::nic@couchbase.com

1001

em::nic.raboy@couchbase.com

1001

uname::nraboy

1001

Compound keys

- Compound keys are look-up documents with a predictable name.
- It's a continuation of the embedded versus referred data discussion.

Compound keys: example

```
u::1001
```

```
 {
    "name": ”Nic Raboy",
    "email": ”nic@couchbase.com",
    "address": ”450 Mission Street",
    "city": ”San Francisco",
    "postCode": ”94105",
    "orders": [ 1, 9, 698, 32 ],
    “productsViewed”: [8, 33, 99, 100]

}
```

Compound keys: example

u::1001

```
 {
    "name": ”Nic Raboy",
    "email": ”nic@couchbase.com",
    "address": ”450 Mission Street",
    "city": ”San Francisco",
    "postCode": ”94105",
    "orders": [ 1, 9, 698, 32 ]
}
```

u::1001::productsviewed

```
{
      "productsList": [
            8, 33, 99, 100
      ]
}
```

Compound keys: example

u::1001

```
{
    "name": ”Nic Raboy",
    "email": ”nic@couchbase.com",
    "address": ”450 Mission Street",
    "city": ”San Francisco",
    "postCode": ”94105",
    "orders": [ 1, 9, 698, 32 ]
}
```

u::1001::productsviewed

```
{
      "productsList": [
            8, 33, 99, 100
      ]
}
```

p::8

```
{
      id": 1,
      "name": "T-shirt",
      "description": "Red Couchbase shirt",
      "quantityInStock": 99,
      "image": "tshirt.jpg”
}
```

Compound keys: example

u::1001

```
{
    "name": ”Nic Raboy",
    "email": ”nic@couchbase.com",
    "address": ”450 Mission Street",
    "city": ”San Francisco",
    "postCode": ”94105",
    "orders": [ 1, 9, 698, 32 ]
}
```

u::1001::productsviewed

```
{
      "productsList": [
            8, 33, 99, 100
      ]
}
```

p::8

```
{
      id": 1,
      "name": "T-shirt",
      "description": "Red Couchbase shirt",
      "quantityInStock": 99
}
```

p::8::img

“http://someurl.com/tshirt.jpg”

Three ways to build a key

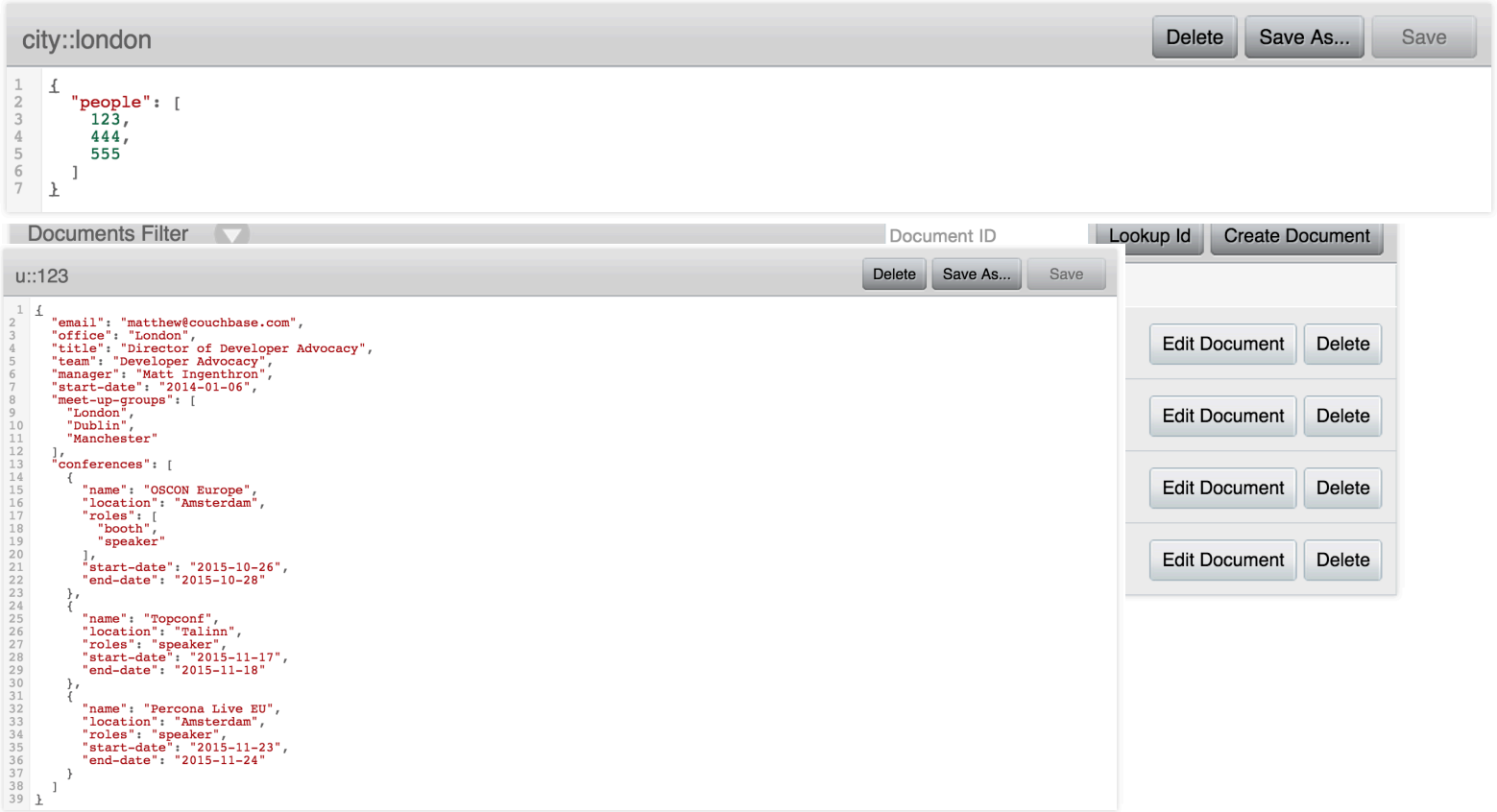
- Human readable/deterministic: e.g. an email address
- Computer generated/random: e.g. UUID
- Compound: e.g. UUID with a deterministic portion

How Do We Query This Data?

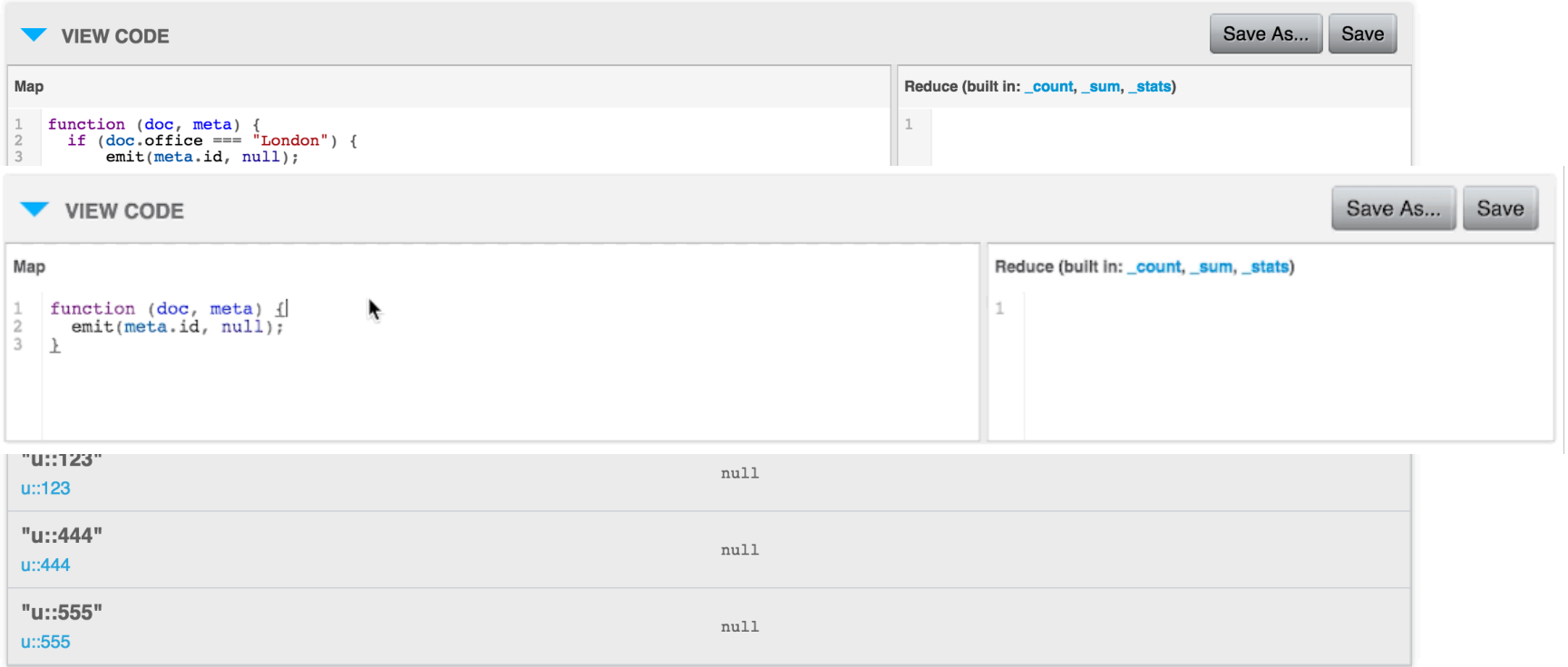
Querying the JSON profile

- Look-up documents: i.e. manual secondary indexing (2i)
- Couchbase views: i.e. automated secondary indexing (2i)
- N1QL

Manual 2i



Automatic 2i: views



Querying With N1QL

What is N1QL?

- **N1QL is a new query language that systematically extends SQL to document-oriented data**

- N1QL extends SQL to handle data that is:
  - **Nested**: Contains nested objects, arrays
  - **Heterogeneous**: Schema-optional, non-uniform
  - **Distributed**: Partitioned across a cluster

Language Highlights

- Standard SELECT pipeline
  - **SELECT, FROM, WHERE, GROUP BY, ORDER BY, LIMIT, OFFSET**
- JOINs (by keys)
- Subqueries
- Aggregation
- Set operators
  - **UNION, INTERSECT, EXCEPT**
- EXPLAIN

Document-oriented Extensions

- Nested
  - Paths—user.profile.email, children[0], children[0:2]
  - **NEST**, **UNNEST**
  - Ranging—**FOR EVERY** child **IN** children
  - Transformations—**SELECT {"name": first_name || last_name}**
- Non-uniform
  - **IS MISSING**
  - Type checking and conversion
- Distributed
  - Direct lookup—**USE KEYS**
  - Efficient joins—**ON KEYS**

Metadata access

Couchbase has metadata about documents in addition to the document values itself.

- The META() function allows for this kind of access

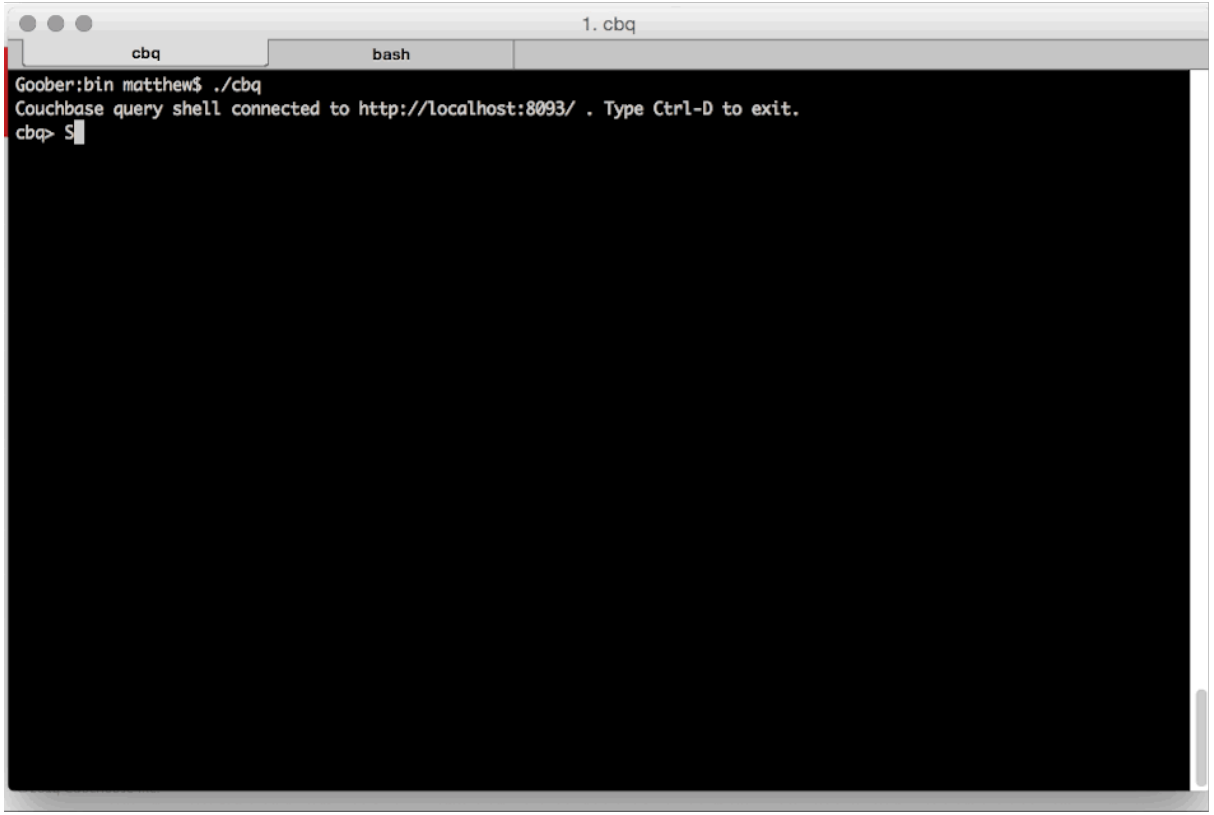
```
select META(`travel-sample`) from `travel-sample` limit 1;          {
            "$1": {
                "cas": 1.60955901476865e+14,
                "flags": 0,
                "id": "landmark_25257",
                "type": "json"
            }
        }
```

Offloading computation to N1QL

N1QL allows you to choose which elements are returned at the top level. This is incredibly useful as it reduces the need to write complex parsing logic:

- Within the application.
- In Client side front end frameworks (Angular/Backbone etc)

N1QL



SELECT

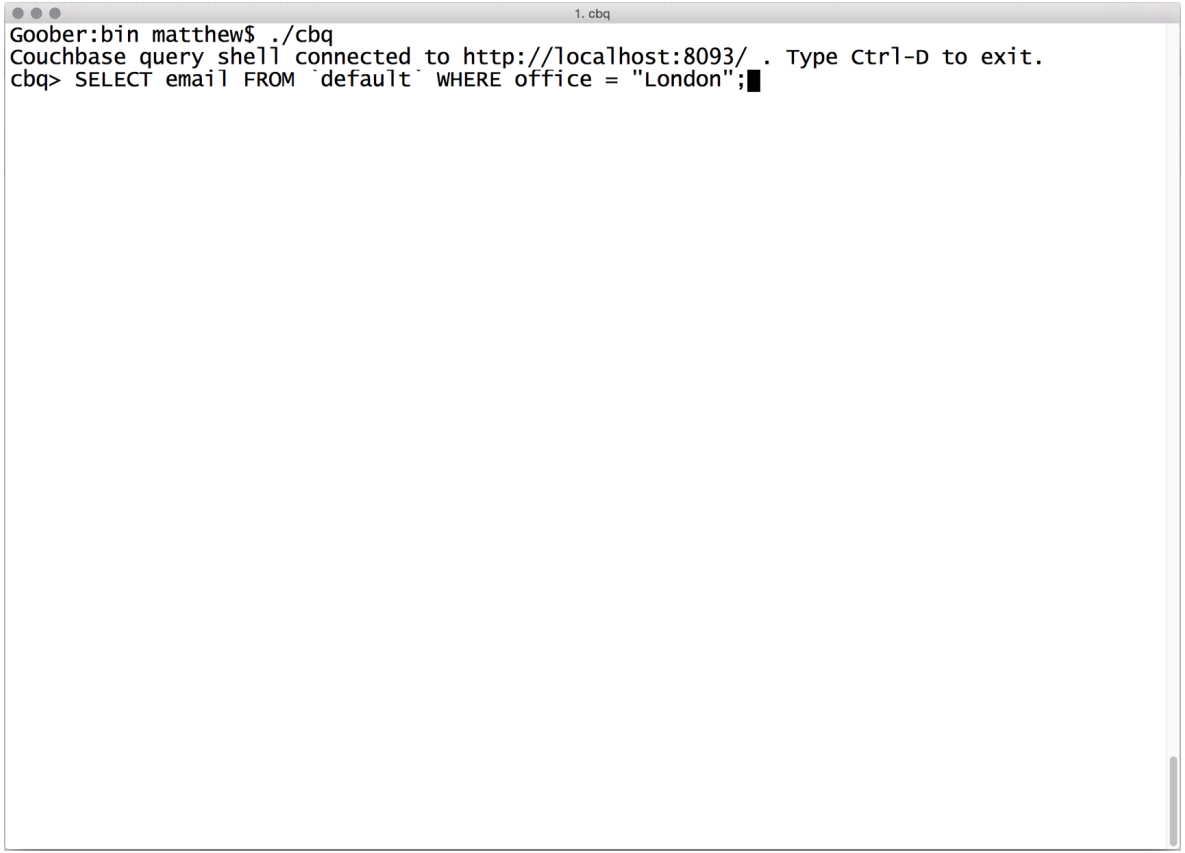
SELECT

```
SELECT COUNT(*)
FROM `default`
WHERE office = "London";
```


```
{
    "requestID": "6e733000-ac83-44ba-95a7-9b012e9c553d",
    "signature": {
        "$1": "number"
    },
    "results": [
        {
            "$1": 6
        }
    ],
    "status": "success",
    "metrics": {
        "elapsedTime": "18.603124ms",
        "executionTime": "18.327696ms",
        "resultCount": 1,
        "resultSize": 31
    }
}
```

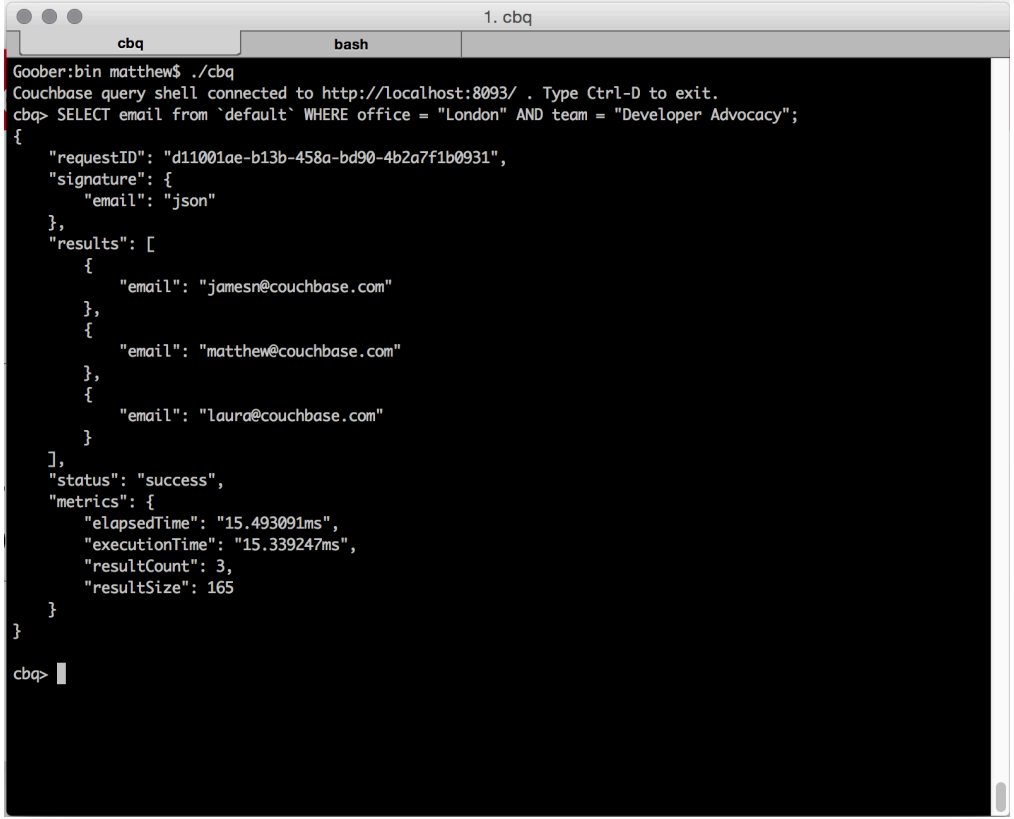
```
SELECT email FROM `default`
WHERE office = "London";
```

SELECT



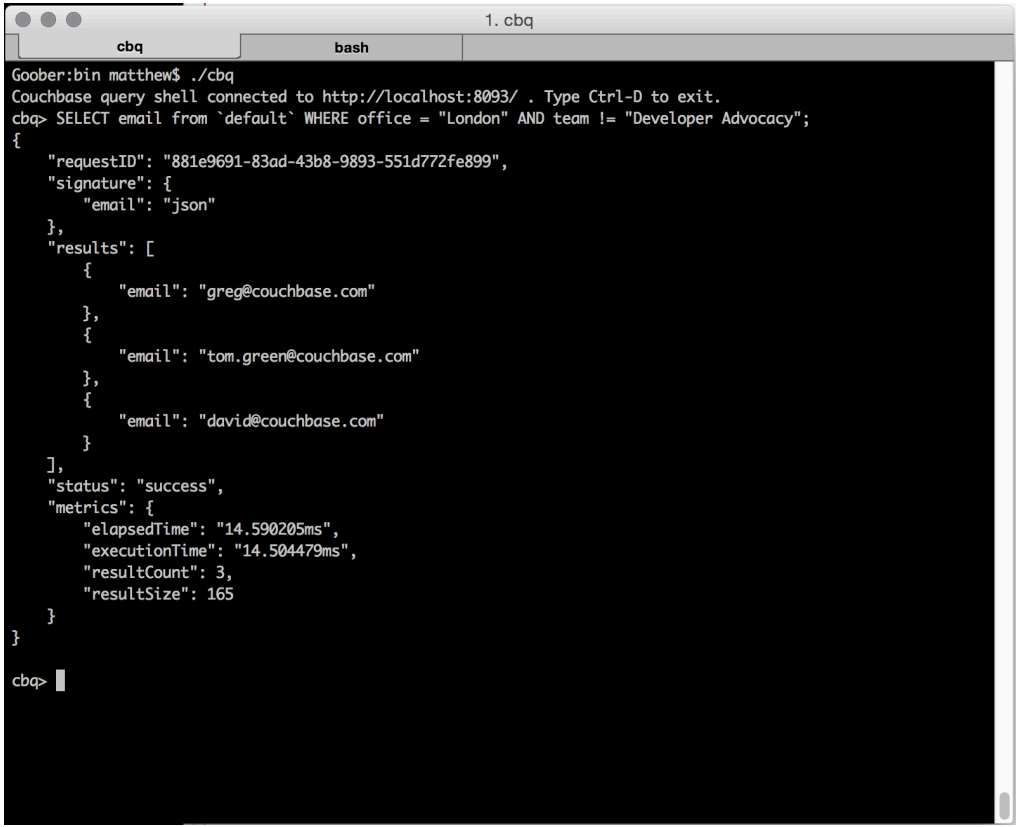
```
SELECT email FROM `default`
WHERE office = "London"
AND team = "Developer Advocacy";
```

SELECT



```
SELECT email FROM `default`
WHERE office = "London"
AND team != "Developer Advocacy";
```

SELECT

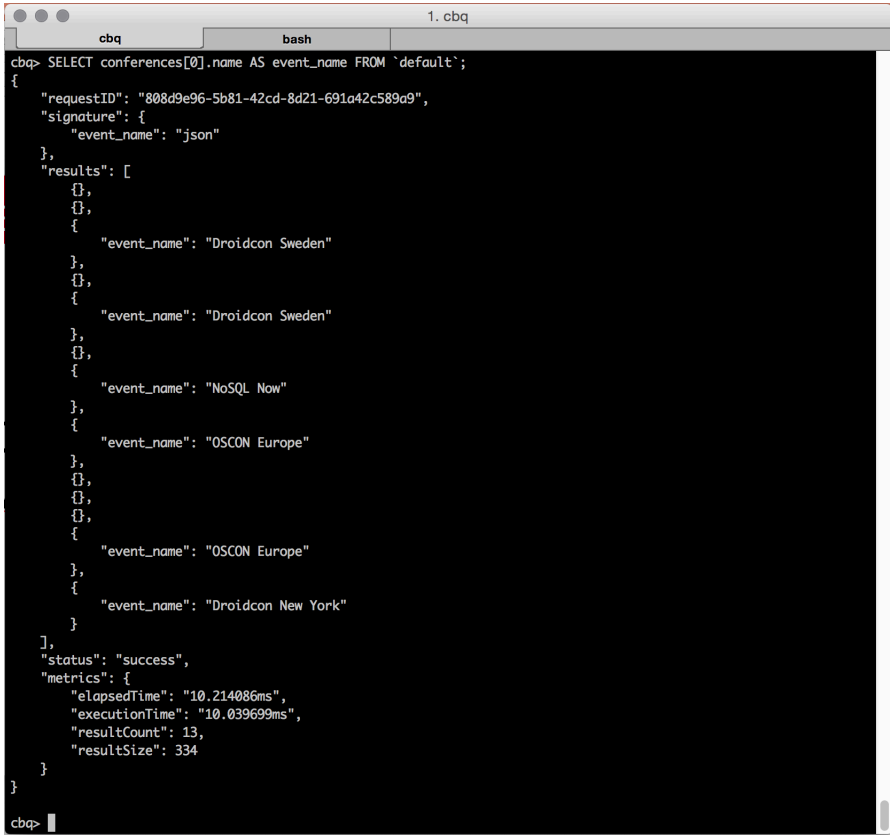


But this is JSON

ARRAY ELEMENTS

```
SELECT conferences[0].name
AS event_name
FROM `default`;
```

ARRAY ELEMENTS



DISTINCT ARRAY ELEMENTS

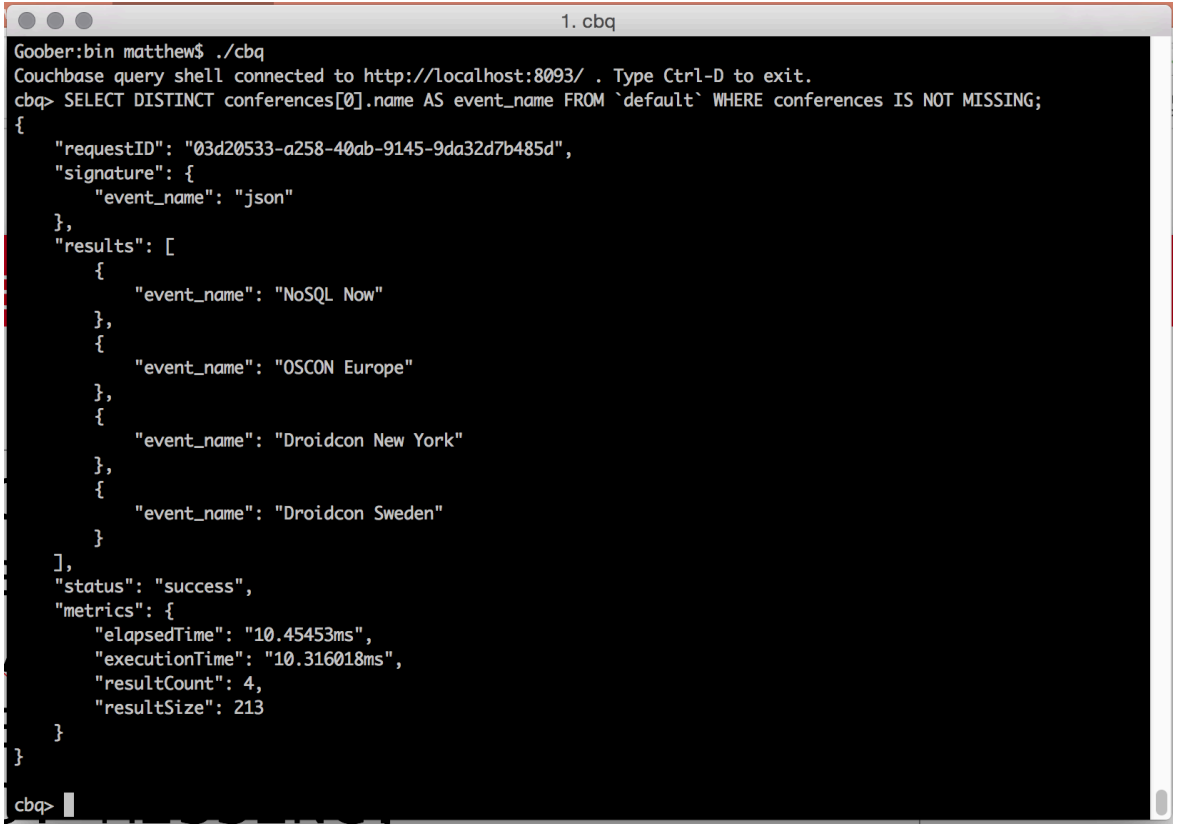
```
SELECT DISTINCT
conferences[0].name
AS event_name FROM `default`;
```

DISTINCT ARRAY ELEMENTS

REMOVE MISSING ITEMS

```
SELECT DISTINCT
conferences[0].name
AS event_name FROM `default`
WHERE conferences
IS NOT MISSING;
```

REMOVE MISSING ITEMS



WHO IS GOING TO DROIDCON SWEDEN?

```
SELECT email AS person,
conferences[0].name AS event
FROM `default`
WHERE ANY event in conferences
SATISFIES event.name =
"Droidcon Sweden" END;
```

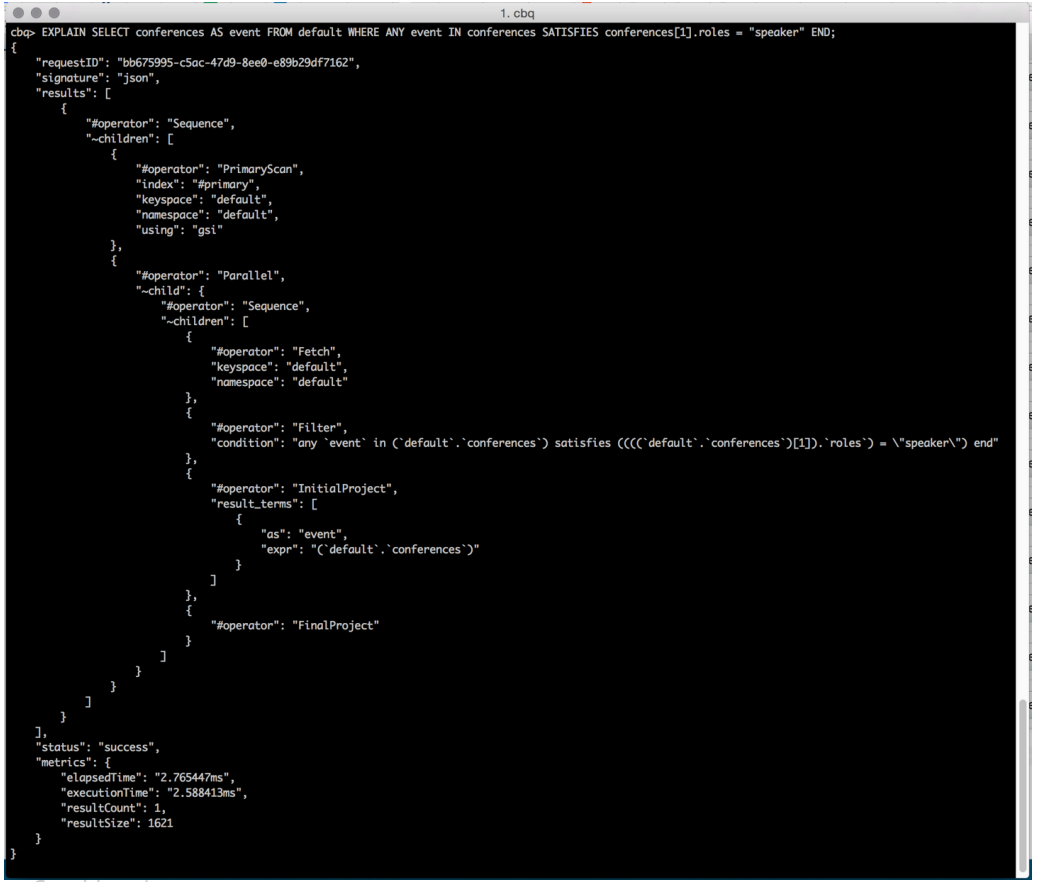
WHO IS GOING TO DROIDCON SWEDEN?

What's going on underneath?

EXPLAIN

```
EXPLAIN SELECT email AS person,
conferences[0].name AS event
FROM `default`
WHERE ANY event in conferences
SATISFIES event.name =
"Droidcon Sweden" END;
```

EXPLAIN



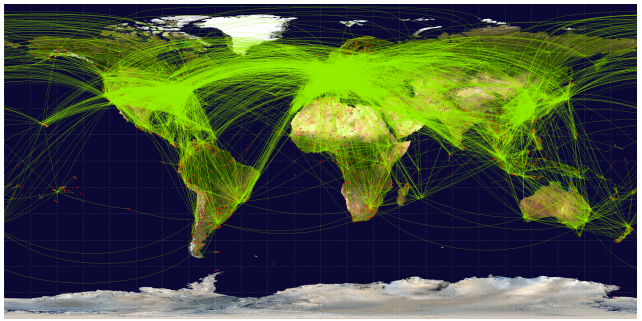
Surely I can write too?

Updating and deleting

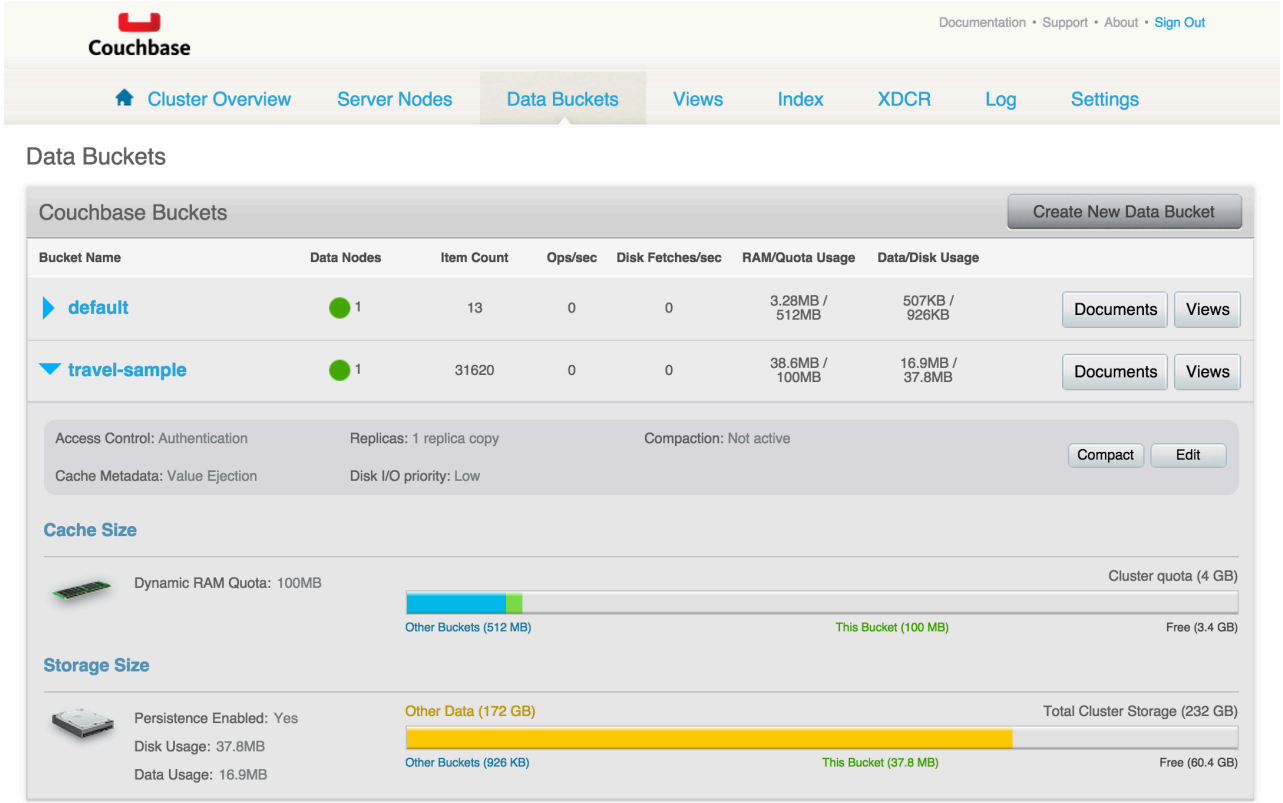
- DELETE: provide the key to delete the document
- INSERT: provide a key and some JSON to create a new document
- UPSERT: as INSERT but will overwrite existing docs
- UPDATE: change individual values inside existing docs

A larger data-set: travel-sample

TRAVEL SAMPLE DATA

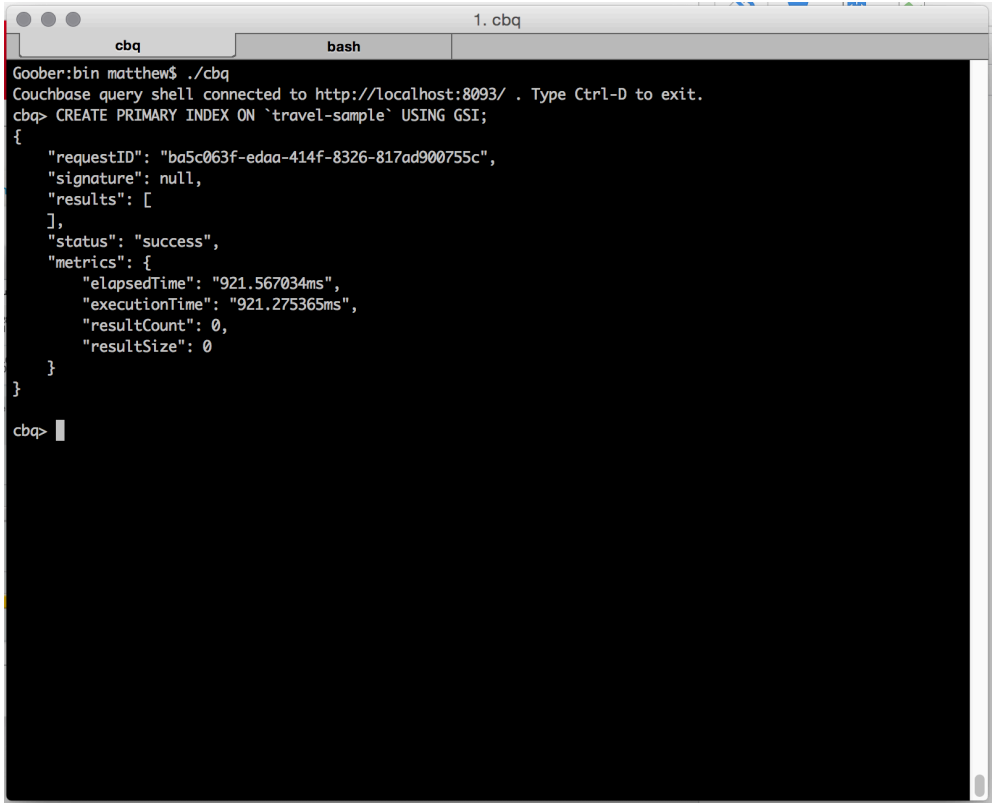


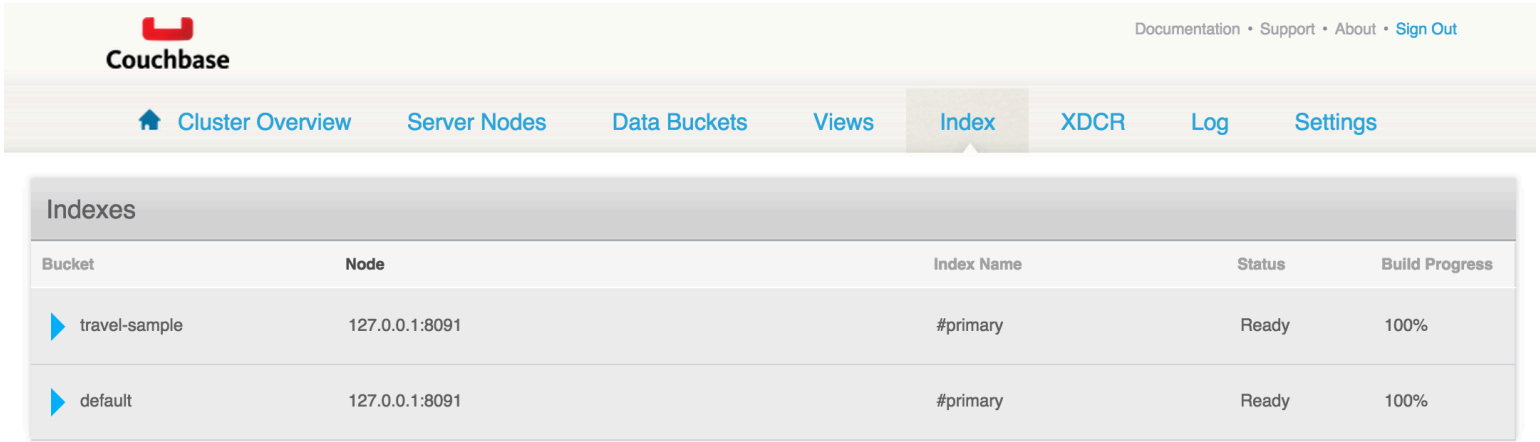
travel-sample



CREATE PRIMARY INDEX

```
CREATE PRIMARY INDEX
ON `travel-sample`
USING GSI;
```

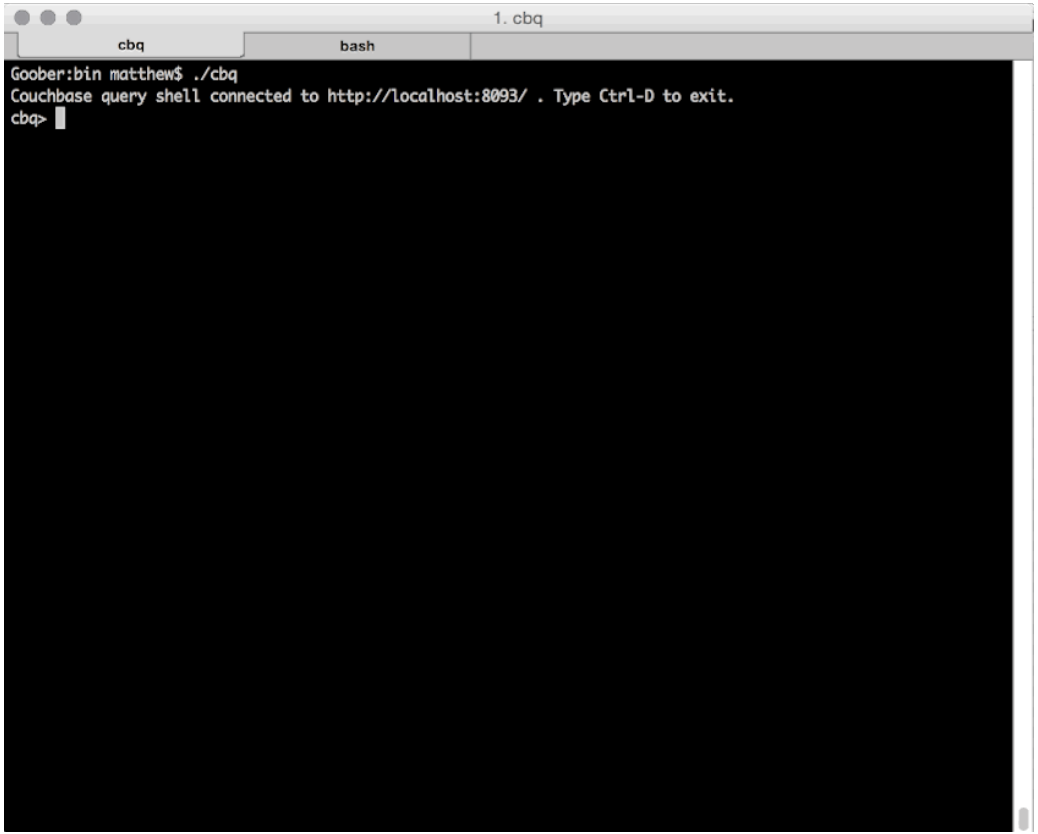


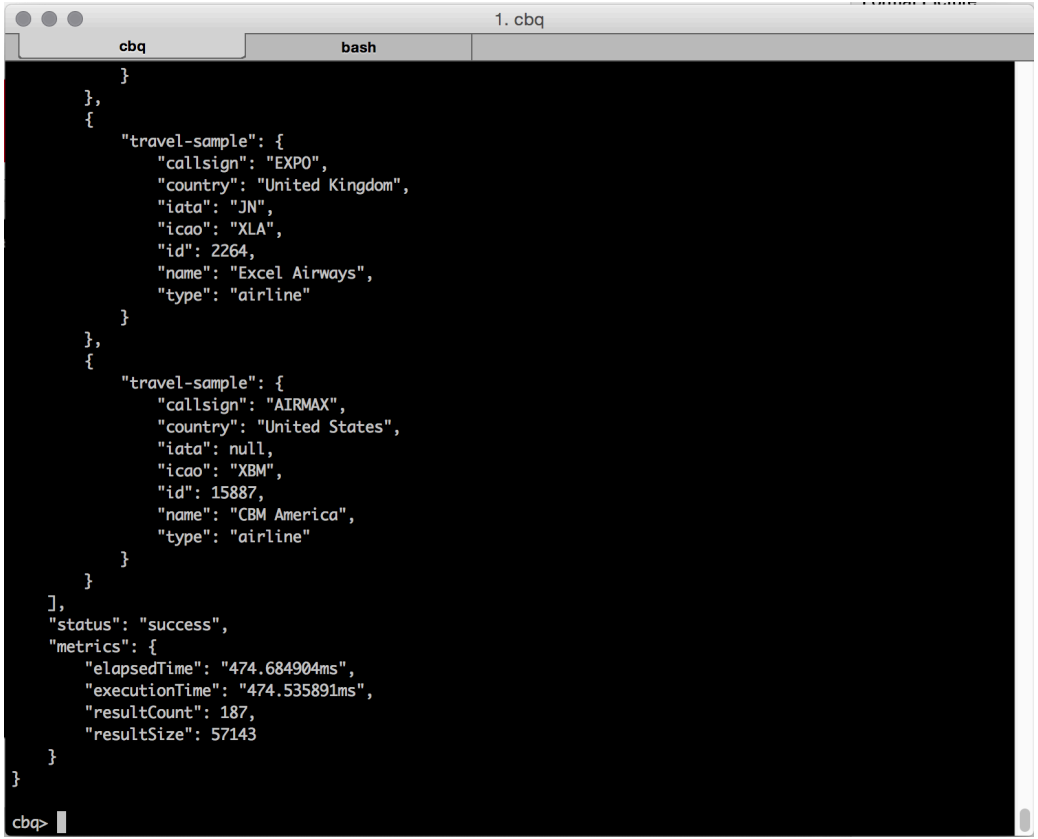


SELECT

```
SELECT * FROM `travel-sample`
WHERE type = "airline";
```

SELECT

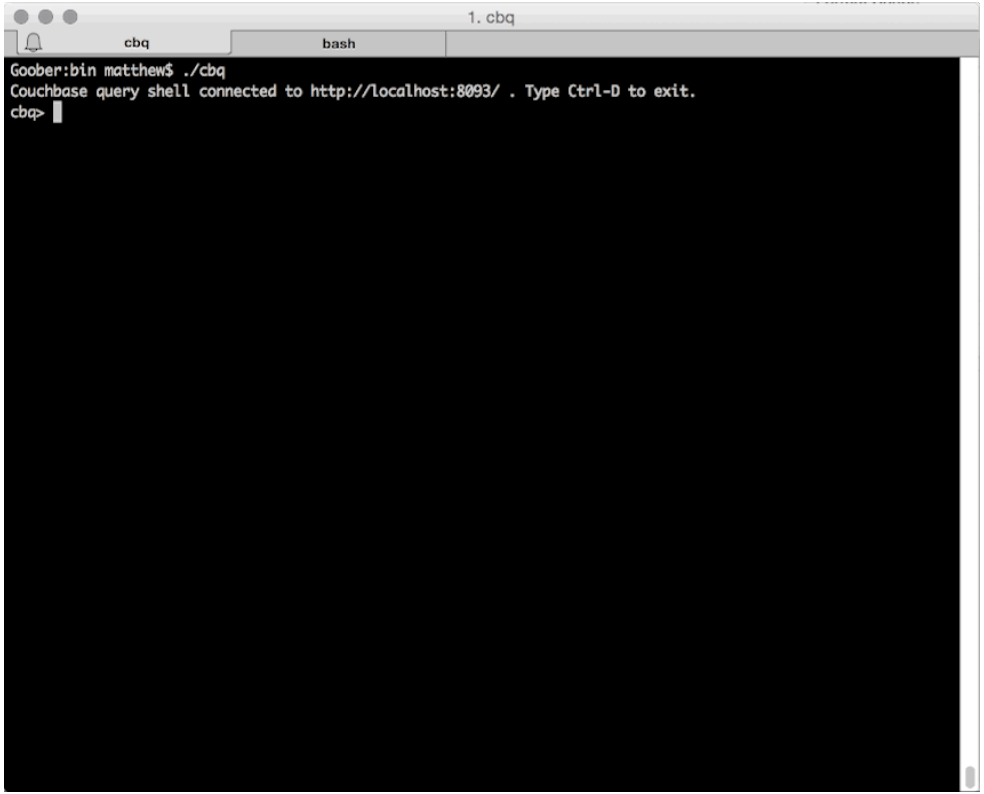


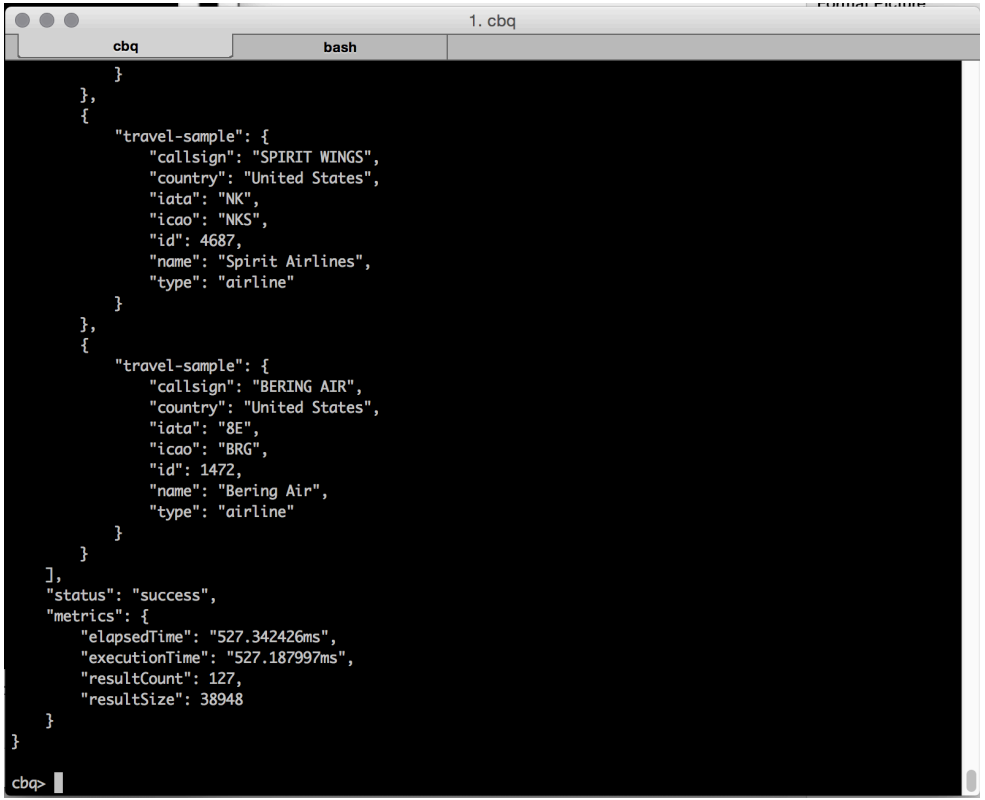


SELECT

```
SELECT * FROM `travel-sample`
WHERE type = "airline"
AND country = "United States";
```

SELECT

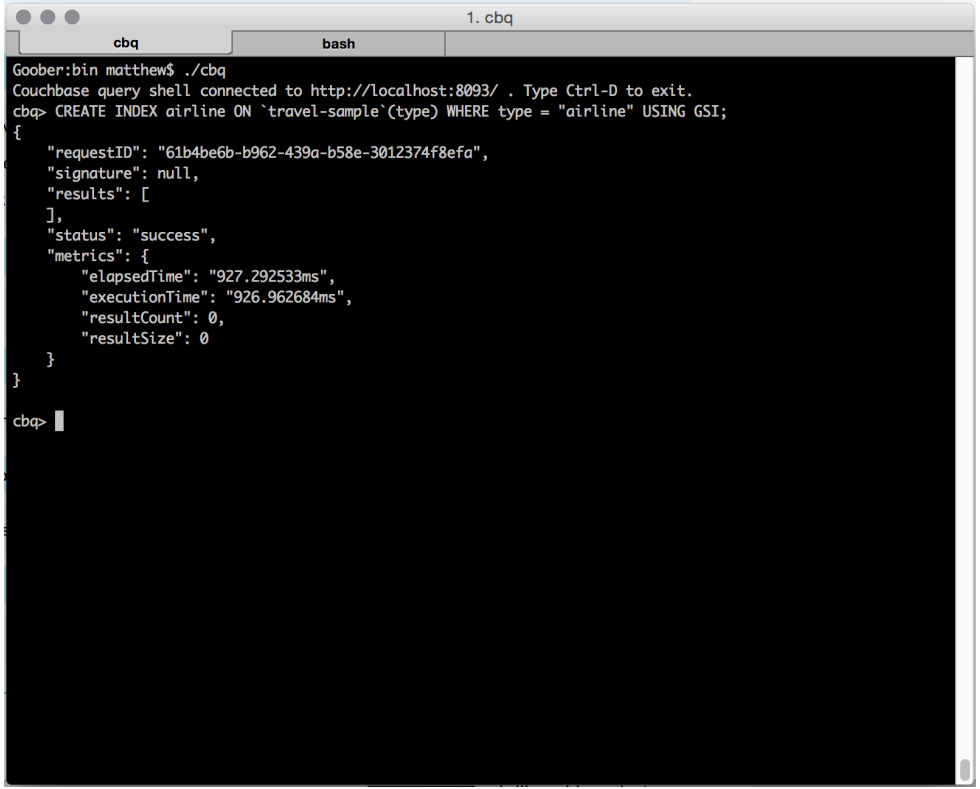


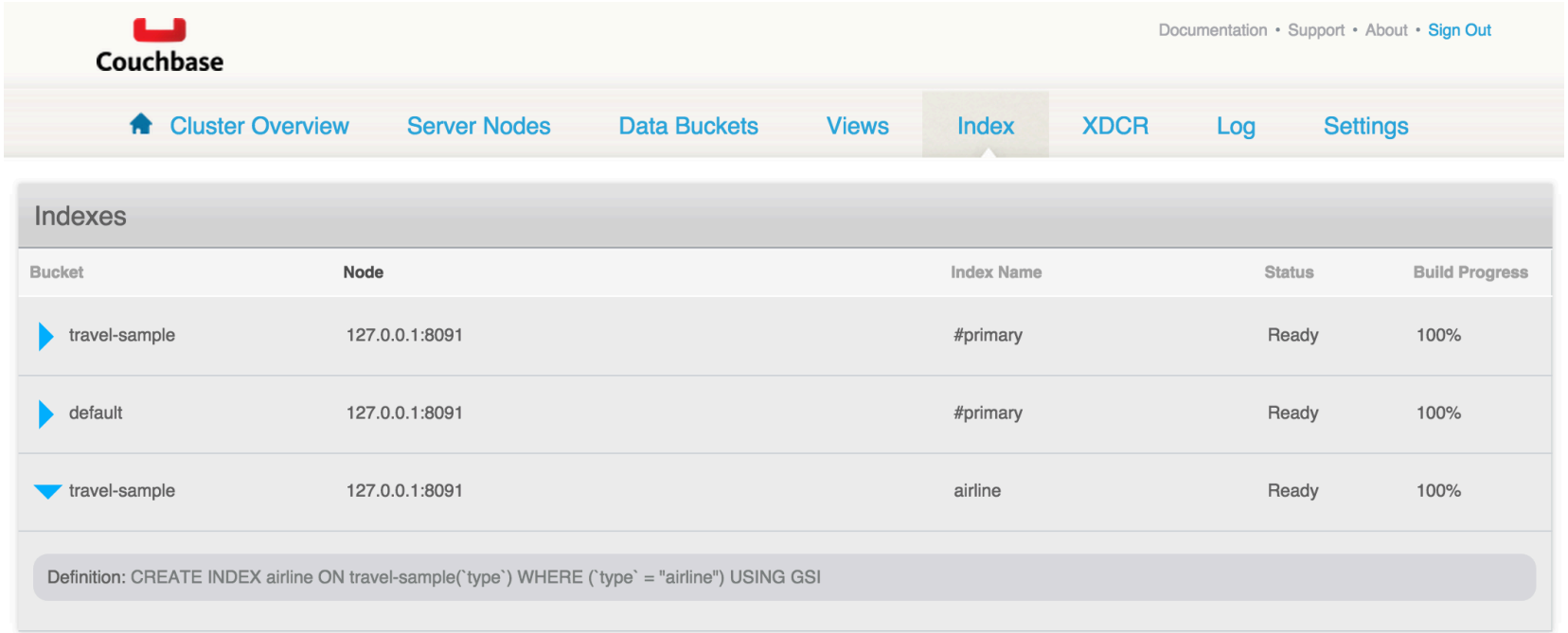


Indexes

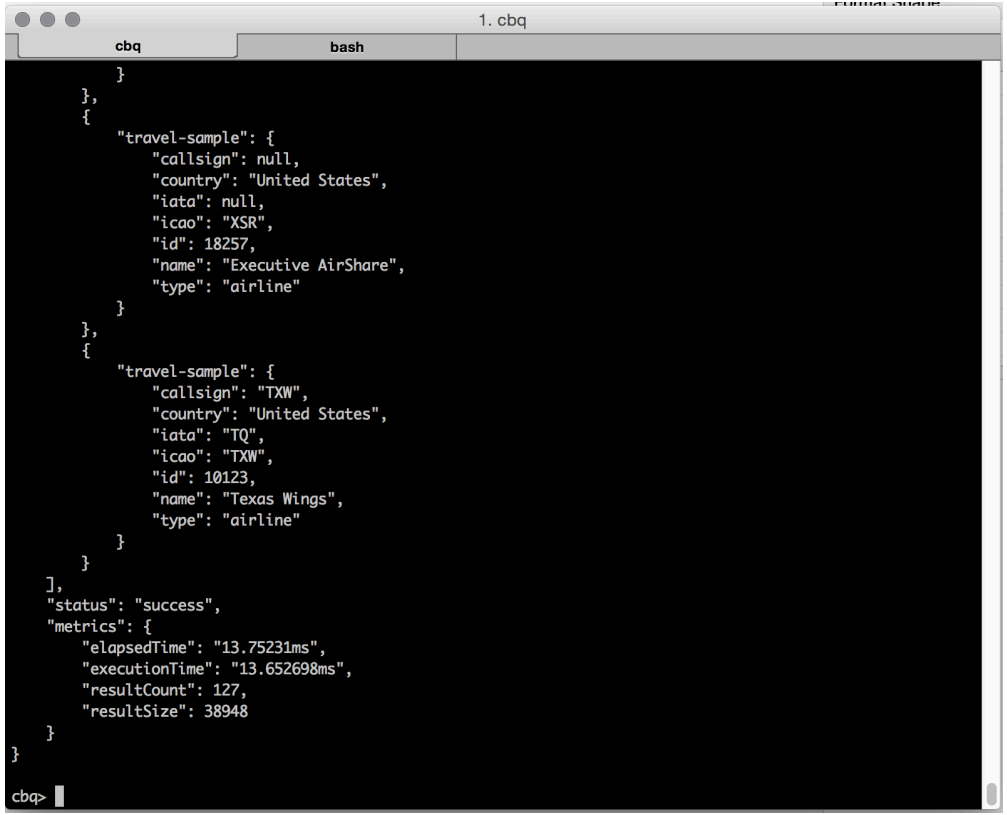
```
CREATE INDEX airline
ON `travel-sample`(type)
WHERE type = "airline"
USING GSI;
```

CREATE INDEX





SELECT WITH AN INDEX



JOINs

JOINs

- Retrieve data from two documents in a single SELECT
- Join within a keyspace/bucket
- Join across keyspaces/buckets

IN JSON

```
{
      "callsign": "UNITED",
      "country": "United States",
      "iata": "UA",
      "icao": "UAL",
      "id": 5209,
      "name": "United Airlines",
      "type": "airline"
}
```

```
{
      "airline": "UA",
      "airlineid": "airline_5209",
      "destinationairport": "SFO",
      "equipment": "777",
      "id": 57047,
      "schedule": [
      {
            "day": 0,
            "flight": "UA894",
            "utc": "02:32:00"        },
            ...
      ],
      "sourceairport": "LHR",
      "stops": 0,
      "type": "route"
}
```

```
{
      "airportname": "Heathrow",
      "city": "London",
      "country": "United Kingdom",
      "faa": "LHR",
      "geo": {
            "alt": 83,
            "lat": 51.4775,
            "lon": -0.461389
      },
      "icao": "EGLL",
      "id": 507,
      "type": "airport",
      "tz": "Europe/London"
}
```

A Simple Join

```
SELECT *
FROM `travel-sample` r
JOIN `travel-sample` a
ON KEYS r.airlineid
WHERE r.sourceairport="LHR"
AND r.destinationairport = "SFO";
```

Who Flies LHR->SFO?

```
SELECT DISTINCT a.name
FROM `travel-sample` r
JOIN `travel-sample` a
ON KEYS r.airlineid
WHERE r.sourceairport="LHR"
AND r.destinationairport = "SFO";
```

Unnesting The Schedule Data

```
SELECT a.name, s.flight, s.utc, r.sourceairport,
r.destinationairport, r.equipment
FROM `travel-sample` r
UNNEST r.schedule s
JOIN `travel-sample` a
ON KEYS r.airlineid
WHERE r.sourceairport="LHR"
AND r.destinationairport = "SFO"
AND s.day=1
ORDER BY s.utc;
```

N1QL Tutorial

http://query.pub.couchbase.com/tutorial

What We've Covered

- Existing NoSQL Databases
- Modeling Document Data
- Designing Document Keys
- Couchbase Document Querying
- N1QL

Demo Time!

A Java SDK Example

Couchbase Labs on GitHub

Node.js
https://github.com/couchbaselabs/try-cb-nodejs

Java
https://github.com/couchbaselabs/try-cb-java

Questions?

We're Hiring!

Thank You!

Nic Raboy
@nraboy (Twitter)