



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?**



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:





nic@couchbase.com

{

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

}



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



London & Developer Advocacy

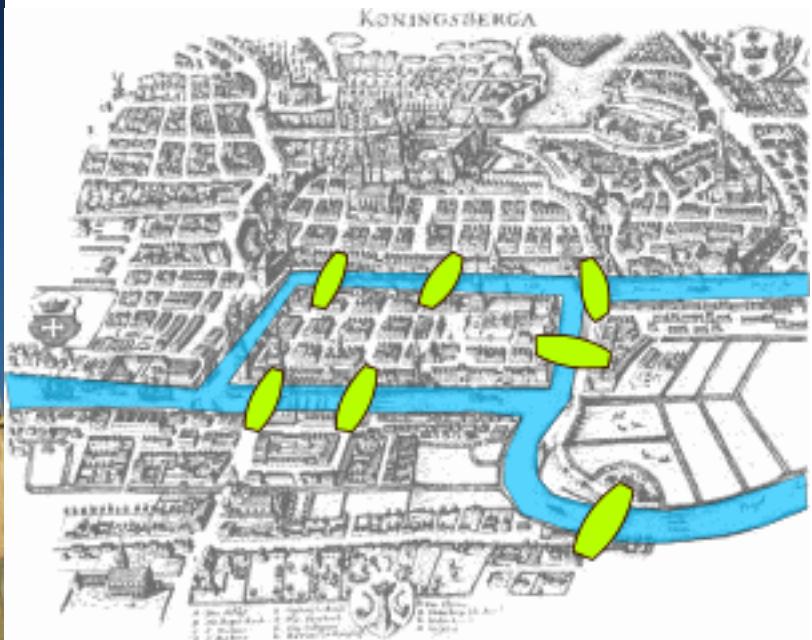
matthew@couchbase.com

laura@couchbase.com

james@couchbase.com

martin@couchbase.com

The Others: Column & Graph

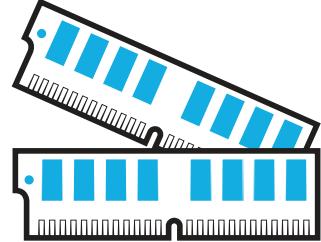


Where Does Couchbase Fit In?

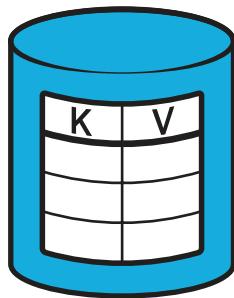
Couchbase provides a complete Data Management solution



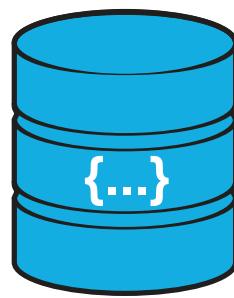
Designed and optimized for interactive web applications



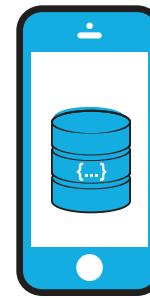
High availability
cache



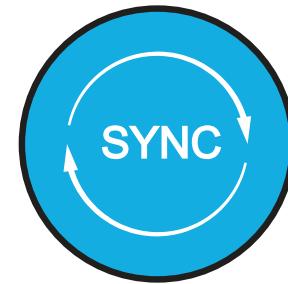
Key-value
store



Document
database



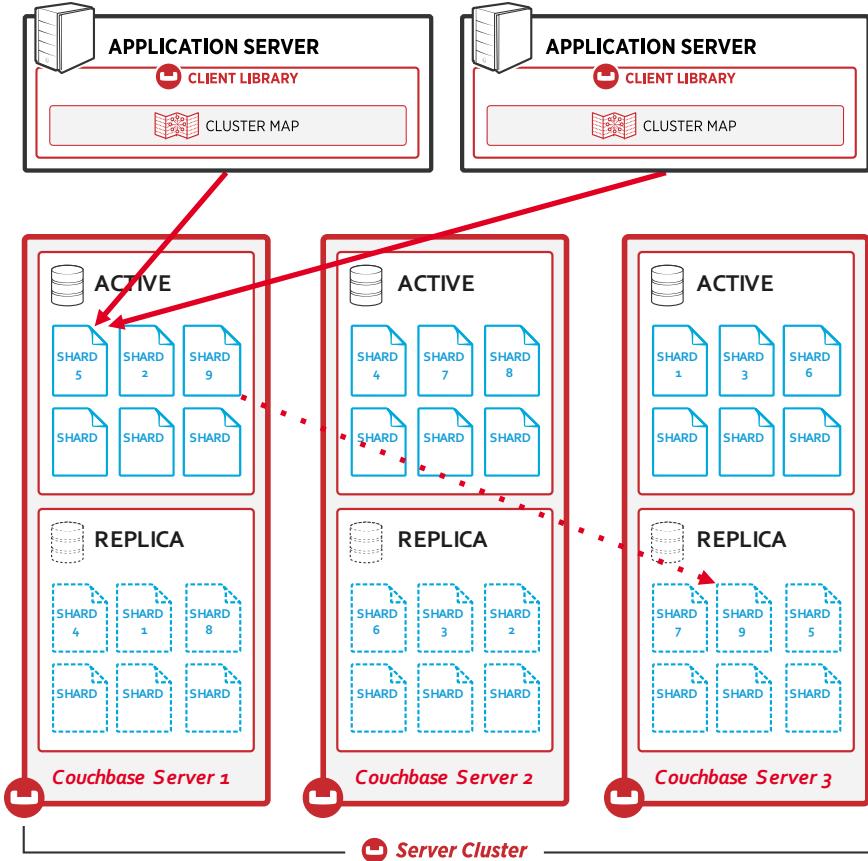
Embedded
database



Sync
management



Basic Operation

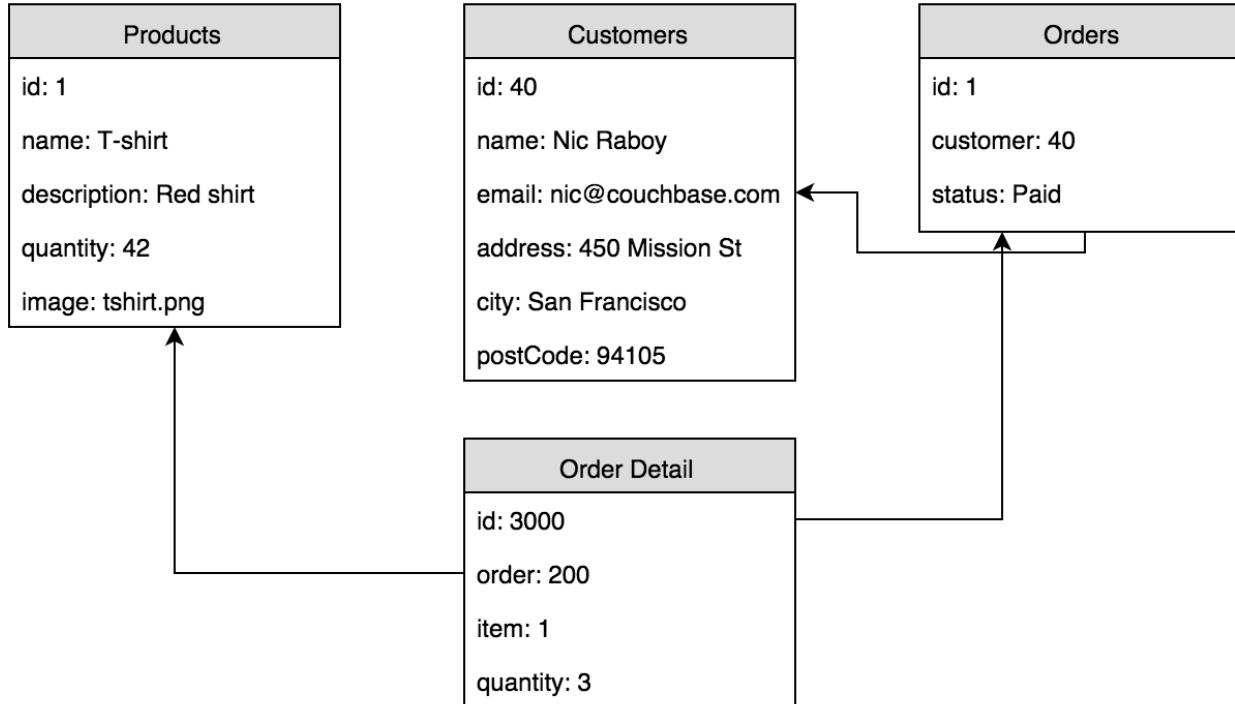


Application has single logical connection to cluster (client object)

- 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

customer-1.json	order-1.json
<pre>1 { 2 "name": "Nic Raboy", 3 "email": "nic@couchbase.com", 4 "address": { 5 "street": "450 Mission St", 6 "city": "San Francisco", 7 "state": "California" 8 }, 9 "company": "Couchbase", 10 "id": 1 11 }</pre>	<pre>1 { 2 "id": 1, 3 "customer": "customer::1", 4 "products": [5 { 6 "product": "product::1", 7 "quantity": 10 8 }, 9 { 10 "product": "product::2", 11 "quantity": 15 12 } 13] 14 }</pre>
<pre>1 { 2 "id": 1, 3 "type": "clothing", 4 "description": "Red Couchbase shirt" 5 }</pre>	



Embedded Document

```
embedded-1.json
1  {
2      "id": 1,
3      "customer": {
4          "name": "Nic Raboy",
5          "email": "nic@couchbase.com",
6          "address": {
7              "street": "450 Mission St",
8              "city": "San Francisco",
9              "state": "California"
10         },
11         "company": "Couchbase"
12     },
13     "products": [
14         {
15             "product": {
16                 "type": "clothing",
17                 "description": "Red Couchbase shirt"
18             },
19             "quantity": 10
20         },
21         {
22             "product": {
23                 "type": "office",
24                 "description": "Couchbase notebook"
25             },
26             "quantity": 15
27         }
28     ]
29 }
```



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 ]  
}
```



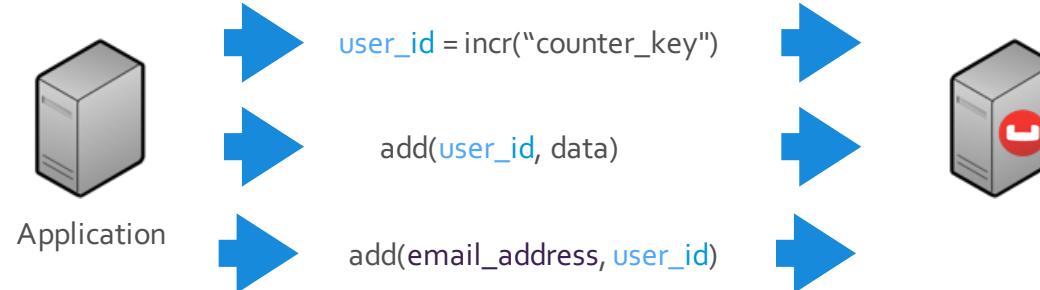
Random/computer generated

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

Creating a new user



Finding user by email address



Multiple look-up documents

	u::count	
.....	1001	fb::16172910
.....	1001	nflx::2939202
.....	1001	twtr::2920283830
.....	1001	em::nic@couchbase.com
.....	1001	em::nic.raboy@couchbase.com
.....	1001	uname::nraboy

```
{ "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": 20jffifieaaixixj }
```



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



city::london

Delete Save As... Save

```
1 {
2   "people": [
3     123,
4     444,
5     555
6   ]
7 }
```

Documents Filter

Document ID

Lookup Id Create Document

u::123

Delete Save As... Save

```
1 {
2   "email": "matthew@couchbase.com",
3   "office": "London",
4   "title": "Director of Developer Advocacy",
5   "team": "Developer Advocacy",
6   "manager": "Mark Ingentron",
7   "start-date": "2014-01-06",
8   "member-groups": [
9     "London",
10    "Dublin",
11    "Manchester"
12  ],
13  "conferences": [
14    {
15      "name": "OSCON Europe",
16      "location": "Amsterdam",
17      "roles": [
18        "booth",
19        "speaker"
20      ],
21      "start-date": "2015-10-26",
22      "end-date": "2015-10-28"
23    },
24    {
25      "name": "popconf",
26      "location": "Tallinn",
27      "roles": "speaker",
28      "start-date": "2015-11-17",
29      "end-date": "2015-11-18"
30    },
31    {
32      "name": "Percona Live EU",
33      "location": "Amsterdam",
34      "roles": "speaker",
35      "start-date": "2015-11-23",
36      "end-date": "2015-11-24"
37    }
38  ]
39 }
```

Edit Document Delete
Edit Document Delete
Edit Document Delete
Edit Document Delete



Automatic 2i: views

VIEW CODE

Map

```
1 function (doc, meta) {  
2   if (doc.office === "London") {  
3     emit(meta.id, null);  
4   }  
5 }
```

Reduce (built in: `_count`, `_sum`, `_stats`)

```
1
```

Save As... Save

VIEW CODE

Map

```
1 function (doc, meta) {  
2   emit(meta.id, null);  
3 }  
4
```

Reduce (built in: `_count`, `_sum`, `_stats`)

```
1
```

Save As... Save

"u::123"	null
u::123	
"u::444"	null
u::444	
"u::555"	null
u::555	

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

A screenshot of a terminal window titled "1. cbq". The window has three tabs at the top: "cbq" (which is active), "bash", and another tab that is mostly obscured. The "cbq" tab displays the following text:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> S
```

The terminal window has a dark background and a light gray border. The text is white or light gray.

SELECT



SELECT

```
1. cbq
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT 1 + 1;
{
  "requestID": "3ccebac7-341a-4c31-a2c5-b46aaed54356",
  "signature": {
    "$1": "number"
  },
  "results": [
    {
      "$1": 2
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "31.826219ms",
    "executionTime": "29.800616ms",
    "resultCount": 1,
    "resultSize": 31
  }
}
cbq> █
```

```
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



The screenshot shows a terminal window with the title "1. cbq". The window contains the following text:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT email FROM `default` WHERE office = "London";■
```

```
1.cbg
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT email FROM `default` WHERE office = "London";
{
  "requestID": "3d5fc31f-7cd9-4a6e-8951-907e533a0dc9",
  "signature": {
    "email": "json"
  },
  "results": [
    {
      "email": "tom.green@couchbase.com"
    },
    {
      "email": "laura@couchbase.com"
    },
    {
      "email": "jamesn@couchbase.com"
    },
    {
      "email": "matthew@couchbase.com"
    },
    {
      "email": "david@couchbase.com"
    },
    {
      "email": "greg@couchbase.com"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "15.412946ms",
    "executionTime": "15.260283ms",
    "resultCount": 6,
    "resultSize": 330
  }
}
cbq> █
```

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

SELECT

The screenshot shows a terminal window with the title "1. cbq". The window has two tabs: "cbq" and "bash". The "cbq" tab is active and displays the following text:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT email from `default` WHERE office = "London" AND team = "Developer Advocacy";
{
  "requestID": "d11001ae-b13b-458a-bd90-4b2a7f1b0931",
  "signature": {
    "email": "json"
  },
  "results": [
    {
      "email": "jamesn@couchbase.com"
    },
    {
      "email": "matthew@couchbase.com"
    },
    {
      "email": "laura@couchbase.com"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "15.493091ms",
    "executionTime": "15.339247ms",
    "resultCount": 3,
    "resultSize": 165
  }
}

cbq> █
```

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

SELECT

The screenshot shows a terminal window with three tabs: 'cbq', 'bash', and '1. cbq'. The '1. cbq' tab is active and displays a JSON response from a Couchbase query. The command run was 'SELECT email from `default` WHERE office = "London" AND team != "Developer Advocacy";'. The output is a JSON object with fields like 'requestID', 'signature', 'results' (containing three email addresses), 'status', and 'metrics' (including execution time and result count).

```
1. cbq
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT email from `default` WHERE office = "London" AND team != "Developer Advocacy";
{
  "requestID": "881e9691-83ad-43b8-9893-551d772fe899",
  "signature": {
    "email": "json"
  },
  "results": [
    {
      "email": "greg@couchbase.com"
    },
    {
      "email": "tom.green@couchbase.com"
    },
    {
      "email": "david@couchbase.com"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "14.590205ms",
    "executionTime": "14.504479ms",
    "resultCount": 3,
    "resultSize": 165
  }
}

cbq> █
```



But this is JSON

ARRAY ELEMENTS



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

ARRAY ELEMENTS



A screenshot of a terminal window titled "1. cbq". The window has two tabs: "cbq" and "bash". The "cbq" tab contains a CBQL query and its results. The query is:

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

The results are:

```
{ "requestID": "808d9e96-5b81-42cd-8d21-691a42c589d9", "signature": { "event_name": "json" }, "results": [ {}, {}, { "event_name": "Droidcon Sweden" }, {}, {}, { "event_name": "Droidcon Sweden" }, {}, {}, { "event_name": "NoSQL Now" }, {}, { "event_name": "OSCON Europe" }, {}, {}, {}, { "event_name": "OSCON Europe" }, {}, { "event_name": "Droidcon New York" } ], "status": "success", "metrics": { "elapsedTime": "10.214086ms", "executionTime": "10.039699ms", "resultCount": 13, "resultSize": 334 } }
```

The "bash" tab is visible at the top but is currently empty.

DISTINCT ARRAY ELEMENTS



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

DISTINCT ARRAY ELEMENTS



1. cbq

cbq bash

```
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT DISTINCT conferences[0].name AS event_name FROM `default`;
{
  "requestID": "55936412-8ec2-472d-afbf-450dca89aa73",
  "signature": {
    "event_name": "json"
  },
  "results": [
    {
      "event_name": "OSCON Europe"
    },
    {
      "event_name": "Droidcon Sweden"
    },
    {
      "event_name": "Droidcon New York"
    },
    {},
    {
      "event_name": "NoSQL Now"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "10.103722ms",
    "executionTime": "9.976059ms",
    "resultCount": 5,
    "resultSize": 215
  }
}
cbq>
```

REMOVE MISSING ITEMS



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

REMOVE MISSING ITEMS



```
1. cbq
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT DISTINCT conferences[0].name AS event_name FROM `default` WHERE conferences IS NOT MISSING;
{
  "requestID": "03d20533-a258-40ab-9145-9da32d7b485d",
  "signature": {
    "event_name": "json"
  },
  "results": [
    {
      "event_name": "NoSQL Now"
    },
    {
      "event_name": "OSCON Europe"
    },
    {
      "event_name": "Droidcon New York"
    },
    {
      "event_name": "Droidcon Sweden"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "10.45453ms",
    "executionTime": "10.316018ms",
    "resultCount": 4,
    "resultSize": 213
  }
}
cbq>
```

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?



```
1. cbq
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> SELECT email AS person,conferences[0].name AS event FROM default WHERE ANY event in conferences SATISFIES event.name = "Droidcon Sweden" END;
{
  "requestID": "d5e99503-c95e-442d-aace-1835b68ee694",
  "signature": {
    "event": "json",
    "person": "json"
  },
  "results": [
    {
      "event": "Droidcon Sweden",
      "person": "jamesn@couchbase.com"
    },
    {
      "event": "Droidcon Sweden",
      "person": "martin@couchbase.com"
    }
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "13.095279ms",
    "executionTime": "12.744933ms",
    "resultCount": 2,
    "resultSize": 192
  }
}
cbq> █
```



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

```
1. cbq
cbq> EXPLAIN SELECT conferences AS event FROM default WHERE ANY event IN conferences SATISFIES conferences[1].roles = "speaker" END;
{
  "requestID": "bb675995-c5ac-47d9-8e0-e89b29df7162",
  "signature": "json",
  "results": [
    {
      "#operator": "Sequence",
      "~children": [
        {
          "#operator": "PrimaryScan",
          "index": "#primary",
          "keyspace": "default",
          "namespace": "default",
          "using": "gsi"
        },
        {
          "#operator": "Parallel",
          "<child": {
            "#operator": "Sequence",
            "~children": [
              {
                "#operator": "Fetch",
                "keyspace": "default",
                "namespace": "default"
              },
              {
                "#operator": "Filter",
                "condition": "any `event` in (`default`.`conferences`) satisfies (((`default`.`conferences`)[1]).`roles`) = \"speaker\") end"
              },
              {
                "#operator": "InitialProject",
                "result_terms": [
                  {
                    "as": "event",
                    "expr": "(`default`.`conferences`)"
                  }
                ]
              },
              {
                "#operator": "FinalProject"
              }
            ]
          }
        ]
      ]
    },
    {
      "status": "success",
      "metrics": {
        "elapsedTime": "2.765447ms",
        "executionTime": "2.588413ms",
        "resultCode": 1,
        "resultSize": 1621
      }
    }
  ]
}
```



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





Couchbase

Documentation • Support • About • Sign Out

Cluster Overview Server Nodes Data Buckets Views Index XDCR Log Settings

Data Buckets

Bucket Name	Data Nodes	Item Count	Ops/sec	Disk Fetches/sec	RAM/Quota Usage	Data/Disk Usage	Actions
▶ default	1	13	0	0	3.28MB / 512MB	507KB / 926KB	Documents Views
▼ travel-sample	1	31620	0	0	38.6MB / 100MB	16.9MB / 37.8MB	Documents Views

Access Control: Authentication Replicas: 1 replica copy Compaction: Not active

Cache Metadata: Value Ejection Disk I/O priority: Low

[Compact](#) [Edit](#)

Cache Size

Dynamic RAM Quota: 100MB Cluster quota (4 GB)

Other Buckets (512 MB) This Bucket (100 MB) Free (3.4 GB)

Storage Size

Persistence Enabled: Yes Total Cluster Storage (232 GB)

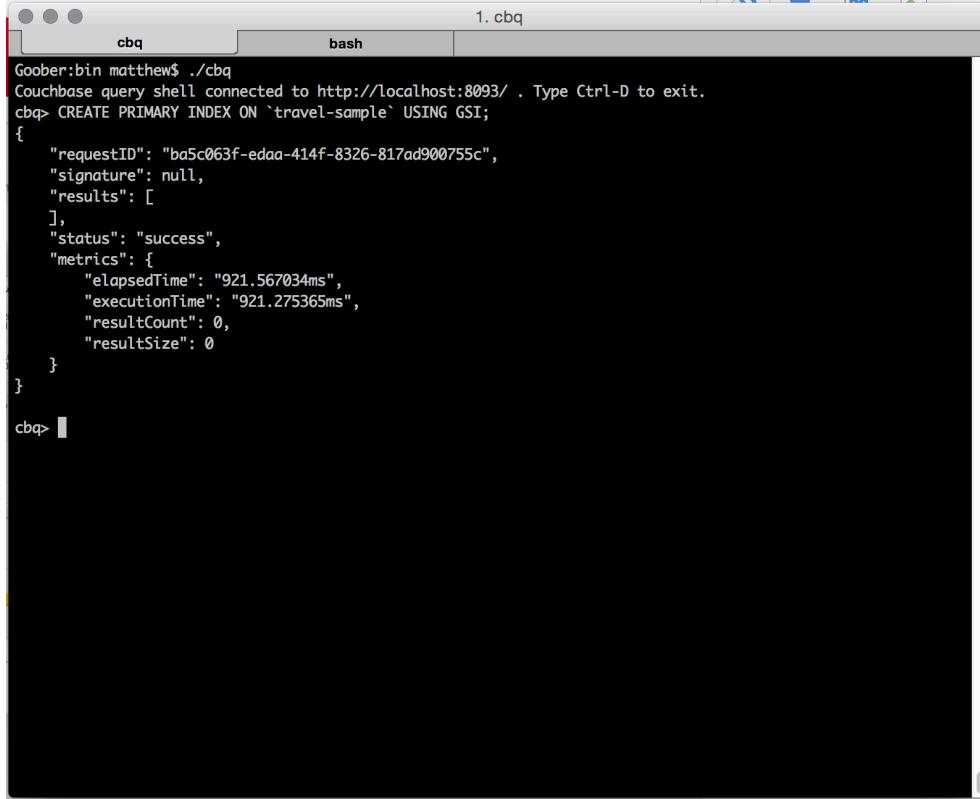
Disk Usage: 37.8MB Other Data (172 GB)

Data Usage: 16.9MB Other Buckets (926 KB) This Bucket (37.8 MB) Free (60.4 GB)

CREATE PRIMARY INDEX



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



The screenshot shows a terminal window with two tabs: 'cbq' (active) and 'bash'. The 'cbq' tab displays the output of a CBQL query. The command 'CREATE PRIMARY INDEX ON `travel-sample` USING GSI;' was run, and the response is a JSON object indicating success:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> CREATE PRIMARY INDEX ON `travel-sample` USING GSI;
{
  "requestID": "ba5c063f-edaa-414f-8326-817ad900755c",
  "signature": null,
  "results": [
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "921.567034ms",
    "executionTime": "921.275365ms",
    "resultCount": 0,
    "resultSize": 0
  }
}
cbq>
```



Couchbase

Documentation • Support • About • [Sign Out](#)

[Cluster Overview](#) [Server Nodes](#) [Data Buckets](#) [Views](#) [Index](#) [XDCR](#) [Log](#) [Settings](#)

Indexes

Bucket	Node	Index Name	Status	Build Progress
▶ travel-sample	127.0.0.1:8091	#primary	Ready	100%
▶ default	127.0.0.1:8091	#primary	Ready	100%

SELECT



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



SELECT

The screenshot shows a terminal window with a title bar "1. cbq". The window has two tabs: "cbq" (which is active) and "bash". The "cbq" tab displays the following text:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> █
```



1. cbq

```
cbq      bash
```

```
        }
    },
{
    "travel-sample": {
        "callsign": "EXPO",
        "country": "United Kingdom",
        "iata": "JN",
        "icao": "XLA",
        "id": 2264,
        "name": "Excel Airways",
        "type": "airline"
    }
},
{
    "travel-sample": {
        "callsign": "AIRMAX",
        "country": "United States",
        "iata": null,
        "icao": "XBM",
        "id": 15887,
        "name": "CBM America",
        "type": "airline"
    }
}
],
"status": "success",
"metrics": {
    "elapsedTime": "474.684904ms",
    "executionTime": "474.535891ms",
    "resultCount": 187,
    "resultSize": 57143
}
}

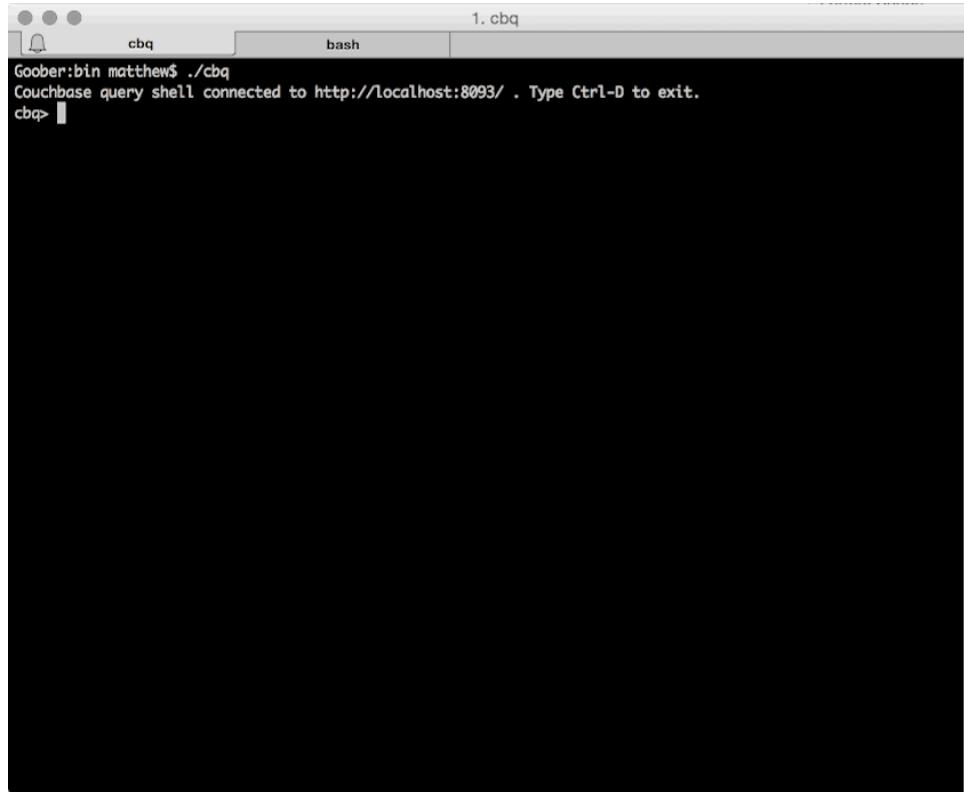
cbq> █
```

SELECT



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

SELECT



A screenshot of a terminal window titled "1. cbq". The window has three tabs at the top: "cbq", "bash", and "bash". The "cbq" tab is active. The terminal output shows:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> 
```

```
1. cbq
cbq> } }
{ "travel-sample": {
    "callsign": "SPIRIT WINGS",
    "country": "United States",
    "iata": "NK",
    "icao": "NKS",
    "id": 4687,
    "name": "Spirit Airlines",
    "type": "airline"
},
{
    "travel-sample": {
        "callsign": "BERING AIR",
        "country": "United States",
        "iata": "8E",
        "icao": "BRG",
        "id": 1472,
        "name": "Bering Air",
        "type": "airline"
    }
},
"status": "success",
"metrics": {
    "elapsedTime": "527.342426ms",
    "executionTime": "527.187997ms",
    "resultCount": 127,
    "resultSize": 38948
}
}
cbq>
```



Indexes

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

CREATE INDEX

The screenshot shows a terminal window titled "1. cbq" with two tabs: "cbq" and "bash". The "cbq" tab is active and displays the following command and its JSON response:

```
Goober:bin matthew$ ./cbq
Couchbase query shell connected to http://localhost:8093/. Type Ctrl-D to exit.
cbq> CREATE INDEX airline ON `travel-sample`(type) WHERE type = "airline" USING GSI;
{
  "requestID": "61b4be6b-b962-439a-b58e-3012374f8efa",
  "signature": null,
  "results": [
  ],
  "status": "success",
  "metrics": {
    "elapsedTime": "927.292533ms",
    "executionTime": "926.962684ms",
    "resultCount": 0,
    "resultSize": 0
  }
}
cbq> █
```



Couchbase

Documentation • Support • About • [Sign Out](#)

[Cluster Overview](#) [Server Nodes](#) [Data Buckets](#) [Views](#) **[Index](#)** [XDCR](#) [Log](#) [Settings](#)

Indexes

Bucket	Node	Index Name	Status	Build Progress
travel-sample	127.0.0.1:8091	#primary	Ready	100%
default	127.0.0.1:8091	#primary	Ready	100%
travel-sample	127.0.0.1:8091	airline	Ready	100%

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

SELECT WITH AN INDEX



The screenshot shows a terminal window titled "1. cbq" with two tabs: "cbq" and "bash". The "cbq" tab contains the following JSON output:

```
{  
    "travel-sample": [  
        {  
            "callsign": null,  
            "country": "United States",  
            "iata": null,  
            "icao": "XSR",  
            "id": 18257,  
            "name": "Executive AirShare",  
            "type": "airline"  
        },  
        {  
            "callsign": "TXW",  
            "country": "United States",  
            "iata": "TQ",  
            "icao": "TXW",  
            "id": 10123,  
            "name": "Texas Wings",  
            "type": "airline"  
        }  
    ],  
    "status": "success",  
    "metrics": {  
        "elapsedTime": "13.75231ms",  
        "executionTime": "13.652698ms",  
        "resultCount": 127,  
        "resultSize": 38948  
    }  
}  
cbq> █
```



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)