

NoSQL Databases Report



Couchbase

Student : Melis HARMANTEPE

Table Of Contents

I. Introduction	3
II. Couchbase Installation on MacOS	4
IV. Create a Bucket and Insert Data	5
a. Create a bucket	5
b. Insert data	6
V. Delete a Bucket or Delete Data	8
a. Drop a bucket	8
b. Delete data	8
VI. Edit a Bucket or a Document	10
VII. Querying	10
VIII. Conclusion	13

I. Introduction

Couchbase Server is an open-source, distributed multi-model NoSQL document-oriented cloud database software package that is optimized for interactive applications. It provides a flexible data model and scalable architecture for storing, retrieving, and querying large amounts of data. It also supports a range of data types and allows for real-time processing and analysis of data, making it suitable for a variety of use cases including web, mobile, and IoT applications. Couchbase also offers features such as built-in caching, indexing, and data replication for high availability and disaster recovery.

Scalability makes it easy to add capacity and handle increasing data volumes without compromising performance. Automatic data replication and failover ensure that the data is always available even in the event of hardware or network failures. In-memory caching can greatly improve the performance of read-heavy workloads. Built-in indexing and querying capabilities make it easy to search and analyze the data. Couchbase provides a lightweight version of its database that can be embedded in mobile applications. This allows for seamless synchronization of data between mobile and backend systems.

The primary database model for Couchbase is document store and the secondary database model is key-value store. Couchbase, provides greater flexibility than relational databases, in that it can store JSON documents with varied schemas. Documents can contain nested structures. This allows developers to express many-to-many relationships without requiring a reference or junction table; and is naturally expressive of hierarchical data.

Overall, Couchbase combines the flexibility of JSON documents, with the speed of in-memory caching and familiarity of an SQL-like query language offering the best NoSQL technology.

Cbq is the commandline interface for couchbase.

Connect to the cbq shell :

```
cd /Applications/Couchbase\ Server.app/Contents/Resources/couchbase-core/bin
```

```
./cbq -u Administrator -p 123456 -engine=http://127.0.0.1:8091
```

```
Meliss-MacBook-Pro:~ silaharmantepe$ cd /Applications/Couchbase\ Server.app/Contents/Resources/couchbase-core/bin
Meliss-MacBook-Pro:bin silaharmantepe$ ./cbq -u Administrator -p 123456 -engine=http://127.0.0.1:8091
Connected to : http://127.0.0.1:8091/. Type Ctrl-D or \QUIT to exit.

Path to history file for the shell : /Users/silaharmantepe/.cbq_history
cbq> █
```

Disconnect from the server, then quit the cbq shell:

```
\DISCONNECT;
\QUIT;
```

I will demonstrate the functionalities of this technology on the **Couchbase web console**.

II. Couchbase Installation on MacOS

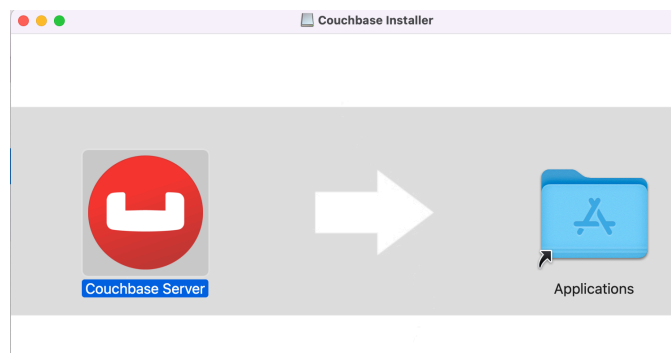
Follow this link : <https://docs.couchbase.com/server/current/install/macos-install.html>

Then from this link : <https://www.couchbase.com/downloads/> select the Couchbase Server for Enterprise download the latest version (7.1.3) for MacOS and click Download.

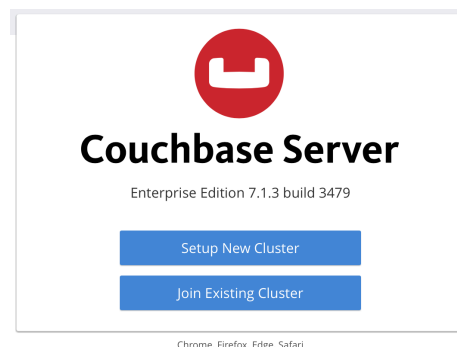
COUCHBASE SERVER 7.1.3 (CURRENT) ENTERPRISE FOR MACOS



Once the installation has ended, drag the Couchbase icon and drop it on the Applications icon of Mac. Go to Applications folder and click on Couchbase to run the server application.



This will open the server on your web browser and the web page will give you 2 options : Create a new cluster or Join existing cluster. Choose the “Create” option since we haven’t created a cluster yet.



Then you will introduce your information and create a password for future logins. I have set the user name as “Administrator” and with that, I chose this password : 123456

From the Applications, click on the Couchbase server icon to run the server. This will open a server on your navigator.

IV. Create a Bucket and Insert Data

a. Create a bucket

In Couchbase, the data is stored within Buckets and Buckets contain documents. Within a Bucket, each document must have a unique key. It is like a database in relational database model. We can have a maximum of 30 buckets in our server. A Bucket provides a container for grouping our data, both in terms of organisation and grouping of similar data and resource allocation. These containers are called Collections and we can have up to 1000 collections within a cluster. Collections are similar to “tables” in RDBMS. For instance, if we have a Bucket that contains travel data, we can have several collections for grouping documents in Airports or Hotels.

Creating a database or a table is equivalent to creating a Bucket in Couchbase server. Each Bucket is assigned a name which is referenced by the user wishing to access the items within it.

RDBMS	Couchbase
Database	Bucket
Table	Collection
Row	Document/Item
Column	Attribute/Field

To create a database, we go to the “Bucket” tab and click on “Add Bucket”. I am creating a table containing the employee information so I will name my Bucket “Employees”. Once I choose a name for our Bucket, I decide the memory to be allocated to this database. I am choosing a small size of memory such as 200 MiB.

Add Data Bucket

Name

authorized users

employees

Bucket Type

☒ Couchbase ☐ Memcached ☐ Ephemeral

Storage Backend

☒ CouchStore ☐ Magma

Memory Quota in megabytes per server node

200

MiB

other buckets (400MiB)

this bucket (200MiB)

available (3.26GiB)

Advanced bucket settings

Replicas

☒ Enable

1

Number of replica (backup) copies

Warning: you do not have enough data servers or server groups to support this number of replicas.

☐ Replicate view indexes

Bucket Max Time-To-Live

☐ Enable

0

seconds

As for the Bucket type, if I were to select the Couchbase of the Ephemeral types, I would be able to enable advanced bucket settings like the number of replication (how many copies we want to create for that bucket), the maximum time-to-live for all the documents in this bucket and “flush” option to remove all items in this bucket.

Flush ⓘ
☒ Enable

Cancel

Add Bucket

In Couchbase server we have three types of Bucket : Couchbase, Ephemeral and Memcached. The most used ones are the Couchbase and the Ephemeral types.

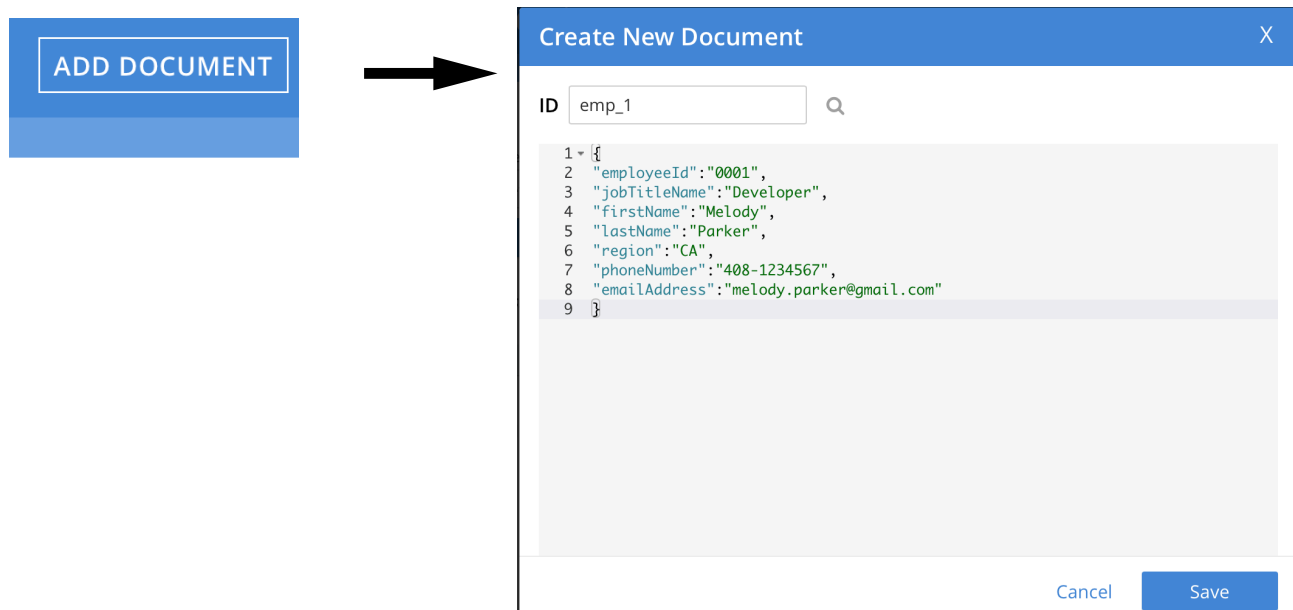
Couchbase buckets allow persistency (storage both in memory/RAM and on disk), replication (for high-availability). If the RAM quota is reached, the data is ejected from the memory but is still kept in the disk. They are designed for use cases that require low latency and high throughput, and can be used to store a wide range of data types, from JSON documents to binary data.

Ephemeral buckets are in-memory buckets that store data only in RAM and don't write to disk. They are designed for use cases that require extremely fast access times and low latency, but do not require data persistence.

Memcached buckets are replicated in-memory buckets that aren't persistent on disk (retained on RAM only) and it reduces the number of queries a database-server must perform. They are designed for use cases that require extremely fast access times and low latency, and use the Memcached protocol for communication with the database.

b. Insert data

To insert data, we must go to the Documents tab and from the top right corner we click on the "Add Document" button. Then we must choose a unique ID for the document that we want to create in order to have access to each document in our bucket. Then we create the data in json format and click Save.



The diagram illustrates the process of adding a new document. On the left, a blue button labeled "ADD DOCUMENT" is shown. An arrow points from this button to a "Create New Document" form on the right. The form has a blue header with the title "Create New Document" and a close button (X). Below the header, there is an "ID" field containing the text "emp_1" and a search icon. The main body of the form contains a JSON document template with the following structure:

```
1 {  
2   "employeeId": "0001",  
3   "jobTitleName": "Developer",  
4   "firstName": "Melody",  
5   "lastName": "Parker",  
6   "region": "CA",  
7   "phoneNumber": "408-1234567",  
8   "emailAddress": "melody.parker@gmail.com"  
9 }
```

At the bottom right of the form, there are "Cancel" and "Save" buttons.

After inserting two other employees to my Employees bucket, we can visualise the contents. Here we have a total of 3 items in our bucket.

Another way of adding data is typing in a query from the query tab :

```
INSERT INTO `Employees` (KEY,VALUE)  
VALUES ( "emp_4",  
        {  
        "employeeId": "0004",  
        "jobTitleName": "Data Scientist",  
        "firstName": "Melis",
```

```

"lastName": "Harmantepe",
"region": "NJ",
"phoneNumber": "403-87250499",
"emailAddress": "melis.harmantepe@gmail.com"
})
RETURNING *;

```

This query adds an employee with the document id “emp_4” into the Employees bucket. We can also add multiple documents in one query. Couchbase is suitable for sparse data, so not every document must have the same fields. There might be data that are missing.

If we have doubts about the new document having the same document id with the other documents in the bucket, we must use the **UPSERT INTO** method. This method replaces the old document’s data with the new values for the document that has the same document id with the one that we want to insert.

```

UPSERT INTO `Employees` (KEY,VALUE)
VALUES ( "emp_4",
{
"employeeId": "0004",
"jobTitleName": "Data Analyst",
"firstName": "Jack",
"lastName": "Nickolson",
"region": "NY",
"phoneNumber": "403-47252329",
"emailAddress": "jack.nickolson@gmail.com"
})
RETURNING *;

```

This query will replace the data of the emp_4 with the new values like “Data Analyst”, “Jack” etc.

Here we can observe that we have successfully added 4 items into our bucket.

The screenshot shows the Couchbase Workbench interface for the 'Employees' bucket. The 'Documents' view displays a table with 4 results. Each document is represented by a key (emp_1, emp_2, emp_3, emp_4) and a JSON value. The documents contain employee information such as employeeId, jobTitleName, firstName, lastName, region, phoneNumber, and emailAddress. The interface also includes a sidebar with navigation options and a top bar with 'project > Documents' and an 'ADD DOCUMENT' button.

id	Value
emp_1	{ "employeeId": "0001", "jobTitleName": "Developer", "firstName": "Melody", "lastName": "Parker", "region": "CA", "phoneNumber": "408-1234567", "emailAddress": "melody.parker@gmail.com" }
emp_2	{ "employeeId": "0002", "jobTitleName": "Software Engineer", "firstName": "Tom", "lastName": "Curry", "region": "NY", "phoneNumber": "422-1038426", "emailAddress": "tom.curry@gmail.com" }
emp_3	{ "employeeId": "0003", "jobTitleName": "Project Manager", "firstName": "Martha", "lastName": "Summer", "region": "CA", "phoneNumber": "401-82377282", "emailAddress": "martha.summer@gmail.com" }
emp_4	{ "employeeId": "0004", "jobTitleName": "Data Analyst", "firstName": "Jack", "lastName": "Nickolson", "region": "NY", "phoneNumber": "403-47252329", "emailAddress": "jack.nickolson@gmail.com" }

The third way is to directly **import a json file** into a bucket. To do this, we click on the “Select file to import” button. We select the json file that we want to upload and we select a Keyspace from the dropdown menu.

Here I imported the rolex.json file into my testBucket :

Workbench

Import

Keyspace

bucket.scope.collection

Limit

Offset

Document ID

show range

N1QL WHERE

testBucket

_default

_default

10

0

optional...













no indexes available...

Retrieve Docs

10 Results for testBucket._default._default, limit: 10, offset: 0

enable field editing

< prev batch | next batch >

	id	
   	002f07bc-f419-4f34-a7e2-baf8fdbd56b3	{"Collection":"Day-Date","Complication":["Day Date"," Date"],"Description":"Diamond Index D Diamond President Bracelet","RRP":91620.0,"Reference":"128239","Size":36}
   	012de1c0-6d4a-4593-a7ed-bd7e96df9bc4	{"Collection":"GMT-Master II","Complication":["Date"," GMT"," Stop Seconds"],"Description":"S Dial","RRP":13620.0,"Reference":"116713LN","Size":40}
   	01613897-2c3a-4e83-9f98-54857505772b	{"Collection":"Datejust","Complication":["Date"," Stop Seconds"],"Description":"Diamond Dial Bracelet","RRP":13920.0,"Reference":"126303","Size":41}

12

Reference": "116500LN", |

Import Data

V. Delete a Bucket or Delete Data

a. Drop a bucket

We might want to delete or replace the integrity of the documents contained in a specific bucket. In this case, we can delete the bucket as a whole. To do this, we go to the Buckets tab and click on the “Drop” button.

Buckets

Backup

XDCR

Security

Settings

Logs

Documents

Query

Indexes

Search

Analytics

Eventing

Views

Employees

4

100%

0

2.7MiB / 200MiB

583KiB

Documents
Scopes & Collections

testBucket

895

100%

0

2.34MiB / 200MiB

3.25MiB

Documents
Scopes & Collections

Type: Couchbase

Bucket RAM Quota: 200MiB

Cluster RAM Quota: 3.84GiB

Replicas: 1

Server Nodes: 1

Ejection Method: Value-Only

Conflict Resolution: Sequence Number

Compaction: Not active

Compression: Passive

Storage Backend: CouchStore

Minimum Durability Level: none

Memory

cluster quota (3.84GiB)

other buckets (400MiB)

this bucket (200MiB)

available (3.26GiB)

Disk

total cluster storage (112GiB)

other buckets (43.5MiB)

this bucket (3.25MiB)

available (71GiB)

Drop

Compact

Edit

b. Delete data

To delete a document from a bucket by key, we can execute this query that deletes the document that has the key “emp_1” and returns the document deleted (RETURNING is optional) :

```
DELETE FROM `Employees` e
USE KEYS "emp_1"
RETURNING e;
```


Query Editor < history (20/20) >

```

1 DELETE FROM `Employees` e
2 USE KEYS "emp_1"
3 RETURNING e

```

Execute
Run as TX
Index Advisor
Explain
✓ success just n
| 300 bytes

Results 📄 🔍 Table JSON

```

1 {
2   {
3     "e": {
4       "emailAddress": "melody.parker@gmail.com",
5       "employeeId": "0001",
6       "firstName": "Melody",
7       "jobTitleName": "Developer",
8       "lastName": "Parker",
9       "phoneNumber": "408-1234567",
10      "region": "CA"
11    }
12  }

```

To delete a document by filter, we can use this example query which filters the data by it's "region" field:

```

DELETE FROM `Employees` e
WHERE e.region = "CA"
RETURNING e.employeeId;

```

Query Editor

```

1 DELETE FROM `Employees` e
2 WHERE e.region = "CA"
3 RETURNING e.employeeId

```

Execute
Run as TX
Index

















Results 📄 🔍

```

1 {
2   {
3     "employeeId": "0003"
4   }
5 }

```

We can also easily delete a specific document from the Documents tab with the delete icon :

	id ▾
   	emp_1
   	emp_2
   	emp_3
   	emp_4

We can remove all of the data from our Bucket with the “**Flush**” button but to use this, we need to enable the “Flush” option when creating a bucket (we can also activate it lateron). This button appears only when it is enabled and deletes all of the documents contained in the bucket but keeps the bucket empty.

Drop
Compact
Flush
Edit

Confirm Flush Bucket

Warning: Bucket data will be lost.
Flush this bucket?

Cancel

Flush Bucket

Keyspace bucket.scope.collection

Employees

_default

Retrieve Docs

No Results for `select meta().id from `Empl``

id

No Results

VI. Edit a Bucket or a Document

We can edit a bucket whenever from the Buckets tab by clicking on the “Edit” button which allow us to reconfigure the bucket. For instance, we can enable the “Flush” option. Once we are done, we click on the “Save changes” and the changes to our bucket are saved.

Buckets

Backup

XDCR

Security

Settings

Logs

cuments

Query

Indexes

Search

Analytics

Eventing

Employees

0

100%

0

2.84MiB / 200MiB

295KiB

Documents

Scopes & Collections

Type: Couchbase

Bucket RAM Quota: 200MiB

Cluster RAM Quota: 3.84GiB

Replicas: 1

Server Nodes: 1

Ejection Method: Value-Only

Conflict Resolution: Sequence Number

Compaction: Not active

Compression: Passive

Storage Backend: CouchStore

Minimum Durability Level: none

Memory

cluster quota (3.84GiB)

other buckets (400MiB)

this bucket (200MiB)

available (3.26GiB)

Disk

total cluster storage (112GiB)

other buckets (46.2MiB)

this bucket (295KiB)

available (71 GiB)

Drop

Compact

Flush

Edit

To edit a document, we use the **UPDATE** method which replaces a document that already exists with updated values.

```
UPDATE `Employees`  
SET region = "New Jersey"  
RETURNING *;
```

This query updates the field “region” with the value “New Jersey” for all of the documents in the bucket Employees.

```
UPDATE `Employees`  
SET region = "Texas"  
WHERE employeeid = "0001"  
RETURNING *;
```

This query updates the region of the employee that has the employeeid of 0001 with the value “Texas”.

VII. Querying

N1QL, the query language used by the Couchbase server, is extremely similar to SQL. We can query JSON data using this strong and expressive query language. Due to its close resemblance to SQL, developers may rapidly become used to it and begin querying Couchbase.

A list of all matching JSON documents in the Bucket is returned when a N1QL query asks for a specific column from the bucket. Unless a WHERE clause is used to filter it out, you will receive a NULL value if the column has no data in the document. We need to activate at least the principal index before we can query a bucket. A secondary index can be included if we wish to increase performance, although it is not required.

Here we see that we can not query the Employees bucket because we haven't created a primary index yet :

The screenshot shows the Couchbase Query Editor interface. On the left, a sidebar contains links for Servers, Buckets, Backup, XDCR, Security, Settings, Logs, Documents, and Query. The main area displays a query: `1 SELECT * FROM `Employees`;`. Below the query are buttons for Execute, Run as TX, Index Advisor, and Explain. The status bar indicates an error: `404` just now | 11.5ms. The Results tab is selected, showing a message: "This query requires an index. A primary index is simple & quick to build (it is also resource-intensive and NOT recommended in production)." with a "Create Primary Index" button and a link to "Learn more about Couchbase indexes".

We create a primary index with this query : **CREATE PRIMARY INDEX primaryindex ON `Employees`;**

The screenshot shows the Couchbase Query Editor with the query: `1 CREATE PRIMARY INDEX primaryindex ON `Employees`;`. The status bar shows "success" just now | 4.1ms | 1 docs | 317 bytes. The Results tab is selected, displaying a JSON response: `{ "results": [] }`.

Now we can start querying. Let's find the employee that has the last name "Nickolson" with this query : **SELECT * FROM `Employees` where lastName = "Nickolson";**

The screenshot shows the Couchbase Query Editor with the query: `1 SELECT * FROM `Employees` where lastName = "Nickolson";`. The status bar shows "success" just now | 4.1ms | 1 docs | 317 bytes. The Results tab is selected, displaying a table with the following data:

emailAddress	employeeid	firstName	jobTitleName	lastName	phoneNumber	region
jack.nickolson@gmail.com	0001	Jack	Data Analyst	Nickolson	403-47252329	Texas

This query finds cities that have a minimum of 400 things to see from the "landmark" collection, using group by and an aggregate function.

**SELECT city City, COUNT(DISTINCT name) LandmarkCount
FROM `travel-sample`.inventory.landmark
GROUP BY city
HAVING COUNT(DISTINCT name) > 400;**

The screenshot shows the Couchbase Query Editor with the query: `1 SELECT city City, COUNT(DISTINCT name) LandmarkCount
2 FROM `travel-sample`.inventory.landmark
3 GROUP BY city
4 HAVING COUNT(DISTINCT name) > 400;`. The status bar shows "success" just now | 131 bytes. The Results tab is selected, displaying a JSON response:

```
1 {  
2   {  
3     "City": "London",  
4     "LandmarkCount": 443  
5   },  
6   {  
7     "City": "San Francisco",  
8     "LandmarkCount": 797  
9   }  
10 }
```

This query lists cities in descending order and then landmarks in ascending order.

```
SELECT city, name  
FROM `travel-sample`.inventory.landmark  
ORDER BY city DESC, name ASC  
LIMIT 5;
```

Query Editor < history (43/43) >

```
1 SELECT city, name
2 FROM `travel-sample`.inventory.landmark
3 ORDER BY city DESC, name ASC
4 LIMIT 5;
```

Execute Run as TX Index Advisor Explain ✓ success just now | 399 bytes

Results Table JSON

```
1 - [
2 - {
3   "city": "Évreux",
4   "name": "Cafe des Arts"
5 },
6 - {
7   "city": "Épinal",
8   "name": "Marché Couvert (covered market)"
9 },
10 - {
```

This query finds the average number of stops per route from the “route” collection with the help of 2 builtin functions (round and avg).

```
SELECT ROUND(AVG(ALL stops), 4)  
AS AvgAllStops FROM `travel-  
sample`.inventory.route;
```

Query Editor < history (41/41) > context unse

```
1 SELECT ROUND(AVG(ALL stops), 4) AS AvgAllStops FROM `travel-sample`.inventory.route;
```

Execute Run as TX Index Advisor Explain ✓ success just now | 603.3ms | 1 docs | 37 bytes

Results Table JSON Chart Plan Plan Text

```
1 - [
2 - {
3   "AvgAllStops": 0.0002
4 }
5 ]
```

This query finds a particular airline based on its Couchbase Server key and returns its metadata as well as the corresponding document, where the key is “airline_10”. The meta() function allows us to return the metadata for a keyspace or document.

```
SELECT meta(al) AS meta, al AS data  
FROM airline al  
WHERE meta(al).id = 'airline_10';
```

Dashboard Servers Buckets Backup XDCR Security Settings Logs Documents Query Indexes Search Analytics Eventing Views

Query Editor ← history (14/23) → travel-sample.inventory

```
1 SELECT meta(al) AS meta, al AS data
2 FROM airline al
3 WHERE meta(al).id = 'airline_10';
```

Execute Explain ✓ success elapsed: 105.01ms | execution: 100.31ms | docs scanned: 1 | docs returned: 1 | size: 244 bytes format ↗

Query Results JSON Table Chart Plan Plan Text

```
1 - [{
2   "meta": {
3     "id": "airline_10",
4     "vbid": 41,
5     "seq": 599,
6     "cas": 1677155705532710912,
7     "flags": 0
8   },
9   "data": {
10    "id": 10,
11    "type": "airline",
12    "name": "40-Mile Air",
```

The Query service is used for operational queries, whereas the Analytics service is used when you don’t know every aspect of the query in advance or when you want to run ad hoc queries for data exploration or visualization.

For more complex, long-running queries, we should use the “Analytics” tab which allows us to do operations like join, set, aggregation, and grouping, which are known to be massive, time-consuming, and memory and CPU resource intensive.

To use the Analytics tab on a Bucket, we must activate the mapping from the top right corner of this tab with the button “Map From Data Service”.

VIII. Conclusion

To conclude, Couchbase Server is an excellent choice for high-performance applications that require fast and responsive data access, real-time analytics, and real-time data processing. Because of its adaptability and scalability, it is a powerful tool for many different scenarios, especially for Data Analytics and Data Processing use cases. For instance, “Query” service is best suited for operational use cases that require real-time data processing, such as e-commerce, gaming, and IoT applications, whereas “Analytics” is best suited for analytical use cases that require complex data analysis and insights, such as business intelligence, machine learning, and fraud detection.

There are several limitations of Couchbase server. First of all, the N1QL query language doesn't offer advanced SQL features such as stored procedures or triggers. Secondly, its data model is limited to a document-based storage, which therefore isn't suitable for more complex relationships between data like graphs. Third, the storage is costly since Couchbase uses an in-memory storage for high performance. Lastly, Couchbase server is relatively new technology so there is still a learning curve associated with using it. It requires more effort to set up and manage.

Couchbase is very similar to MongoDB which is also an open-source, document-oriented distributed database. Couchbase and MongoDB offer both built-in indexing and geospatial queries. They are both scalable and have a flexible data model. However, MongoDB is mostly horizontally scalable, so it is more limited in terms of scalability. Moreover, MongoDB's query language is far more complex than N1QL language.

Couchbase and Cassandra have both flexible data model, are designed for horizontal scalability and support strong consistency. They are also designed for high performance and can handle large volumes of data with low latency. However, Cassandra is a column-oriented distributed database and Couchbase is document-oriented.

Couchbase and Redis are both horizontally scalable and are designed for high performance. Couchbase offers high consistency but Redis can have some inconsistencies between different nodes in the cluster. Redis has a larger community and a more extensive ecosystem. Compared to Redis, Couchbase has a higher learning curve and requires more effort to set up and manage. Also, Redis is key-value-oriented.

Couchbase differs a lot from the Neo4j technology. First of all, Neo4j is a graph database that is optimized for handling complex relationships between data and its scaling mechanisms are more limited than Couchbase. However, both Neo4j and Couchbase offer strong consistency. Neo4j's performance is more limited compared to Couchbase, and its community is more focused on graph databases.