

JAVA DEVELOPER WORKSHOP

Manuel Hurtado

Solutions Engineer at Couchbase





AGENDA

Lab Environment

Introduction to Data Modeling

Travel-sample data model

Connecting to Couchbase

Key/value and CRUD

Working with Parts of a Document

Querying for Documents with N1QL

Batching with RXJava

Full Text Search

Spring Boot



LAB ENVIRONMENT





Lab 0: Installing Couchbase

DOWNLOAD

<https://www.couchbase.com/downloads>

DEPLOYMENT OPTIONS

<https://developer.couchbase.com/documentation/server/current/install/get-started.html>

 Non-cloud On-Premises	 Virtualized/Containerized Environment	 Public Cloud
Run Couchbase within your bare metal data center.	Run Couchbase within a virtualized or containerized environment such as VMware or Docker.	Deploy and manage Couchbase in a public cloud such as Amazon Web Services(AWS), Microsoft Azure, or Google Cloud Platform.

Lab



Lab 0: Installing Couchbase

DOCKER

- Documentation: <https://developer.couchbase.com/documentation/server/current/install/docker-deploy-single-node-cluster.html>
- Single server deployment for this lab
- TIP: create local directory and map to enable persistence upon recreating containers

```
docker run -d -p 8091-8094:8091-8094 -p 11210:11210 --name cb1 -v  
/[FULL_PATH]/cb1:/opt/couchbase/var couchbase
```

Lab



Lab 1: Initial Setup

Access UI Web Console: <http://localhost:8091/ui/index.html>

Enter Cluster Name

Administration User: Administrator:password

A screenshot of a web browser window showing the 'New Cluster' setup page for Couchbase. The URL in the address bar is 'localhost:8091/ui/index.html'. The page has a blue header bar with the Couchbase logo and the text 'Couchbase > New Cluster'. Below the header are four input fields: 'Cluster Name' (containing 'CB5 Docker'), 'Create Admin Username' (containing 'Administrator'), 'Create Password' (containing '*****'), and 'Confirm Password' (containing '*****'). At the bottom left is a 'Back' button, and at the bottom right is a blue 'Next: Accept Terms' button.

localhost:8091/ui/index.html

HipChat ToRead Couchbase VECI Varios GitTalent

Couchbase > New Cluster

Cluster Name
CB5 Docker

Create Admin Username
Administrator

Create Password

Confirm Password

< Back

Next: Accept Terms

Lab



Lab 1: Initial Setup

Memory Quotas:

- Data 1024 Mb
- Index: 246 Mb
- Search: 256 Mb

Couchbase > New Cluster / Configure

Host Name / IP Address Usually localhost or similar
127.0.0.1

Data Disk Path Path cannot be changed after setup
/opt/couchbase/var/lib/couchbase/data
Free: 195 GB

Indexes Disk Path Path cannot be changed after setup
/opt/couchbase/var/lib/couchbase/data
Free: 195 GB

Couchbase Memory Quotas Per service / per node

<input checked="" type="checkbox"/> Data Service	1024	MB
<input checked="" type="checkbox"/> Index Service	512	MB
<input checked="" type="checkbox"/> Search Service	256	MB
<input checked="" type="checkbox"/> Query Service	-----	

TOTAL QUOTA 1792MB

RAM Available 5957MB Max Allowed Quota 4933MB

Index Storage Setting
 Standard Global Secondary
 Memory-Optimized ⓘ

Enable software update notifications in the web console

Lab



Lab 1: Initial Setup

Install sample data set

Install travel-sample data set: Admin Console > Settings > Sample Buckets

The screenshot shows the 'CB5 Docker > Settings' page in the Couchbase Admin Console. The URL in the browser bar is `localhost:8091/ui/index.html#/settings/sampleBuckets`. The top navigation bar includes links for HipChat, ToRead, Couchbase, VECI, Varios, and GitTalent, along with Activity, Documentation, Support, and Administrator options.

The main content area has a title 'CB5 Docker > Settings' with a back button icon. Below it is a horizontal navigation bar with tabs: Cluster, Software Updates, Auto-Failover, Email Alerts, Auto-Compaction, and Sample Buckets (with a dropdown arrow). The 'Sample Buckets' tab is currently selected.

A message states: 'Sample buckets contain example data and Couchbase views. You can provision one or more sample buckets to help you discover the power of Couchbase Server.' A note in red text below says: 'Sample buckets (like all buckets in Couchbase Server 5.0+) can only be accessed by a user with privileges for that bucket.'

Under 'Available Samples', there are three checkboxes: 'beer-sample', 'gamesim-sample', and 'travel-sample'. The 'travel-sample' checkbox is checked. Below these is a blue 'Load Sample Data' button.

Under 'Installed Samples', it says 'none'.

Lab

Lab 1: Initial Setup

Create User



- Username: travel
- Password: password
- Roles:
 - Bucket Full Access[travel-sample]
 - Query Update[travel-sample]
 - Query Select[travel-sample]
 - Query Manage Index[travel-sample]
 - Query Insert[travel-sample]
 - Query Delete[travel-sample]

Add New User

Authentication Domain
 Couchbase External

Username
travel

Full Name (optional)
travel

Password
.....

Verify Password
.....

Roles

- Admin
- Cluster Admin
- Read Only Admin
- Bucket Roles
 - ▶ Bucket Admin
 - ▶ Bucket Full Access
 - all [*] ⓘ
 - travel-sample ✓
 - ▶ Data Roles
 - ▶ FTS Roles
 - ▶ Query Roles

Cancel Save

CB5 Docker > Security

Users ▼ Root Certificate Audit

username ▼	full name	roles	auth domain
travel	travel	Query Update[travel-sample], Query Select[travel-sample], Query Select[sample], Query Manage Index[travel-sample], Query Insert[travel-sample], Query Delete[travel-sample], Bucket Full Access[travel-sample]	Couchbase

Lab

Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2017. All rights reserved.

9

Lab 1: Initial Setup

Clone repo



Download the workshop code from this repository:

```
git clone https://github.com/mahurtado/CouchbaseJavaWorkshop.git
```

Build a simple executable jar:

```
cd CouchbaseJavaWorkshop  
mvn clean compile assembly:single
```

Run main class:

```
java -classpath target/CbDevWorkshop-0.0.1-SNAPSHOT-jar-with-dependencies.jar -  
Dcbworkshop.clusteraddress=localhost -Dcbworkshop.user=travel -  
Dcbworkshop.password=password -Dcbworkshop.bucket=travel-sample com.cbworkshop.MainLab
```

Lab



INTRODUCTION TO DATA MODELING





The Plan – Introduction to Data Modeling

- Designing a Document Key
- Embedding Data as Complex JSON in Documents
- Create References Between Documents



Building a Document 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 Document Keys

```
key: nic@couchbase.com
```

```
{
  "name": "Nic Raboy",
  "email": "nic@couchbase.com",
  "address": "450 Mission Street",
  "city": "San Francisco",
  "state": "CA"
}
```



Generated Document Keys

```
key: f8d8d616-6ce0-4344-9793-4ec5a676c223
```

```
{
  "name": "Nic Raboy",
  "email": "nic@couchbase.com",
  "address": "450 Mission Street",
  "city": "San Francisco",
  "state": "CA"
}
```



Compound Document Keys

```
key: profile::f8d8d616-6ce0-4344-9793-4ec5a676c223
```

```
{  
    "name": "Nic Raboy",  
    "email": "nic@couchbase.com",  
    "address": "450 Mission Street",  
    "city": "San Francisco",  
    "state": "CA"  
}
```



Counter Document Keys

```
key: msg::66565
```

```
{
  "date": "2018-01-02 23:55:55",
  "user": "nic@couchbase.com",
  "msg": "This is a test message"
}
```

Counters documentation:

<https://developer.couchbase.com/documentation/server/5.0/sdk/core-operations.html#story-h2-5>

```
// Java
// Create the counter
bucket.counter("myCounter", 0, 20);
// Use counter for new key
String id = "msg::" + bucket.counter(("myCounter", 1).content());
```



Mixed Document Key Example

```
key: f8d8d616-6ce0-4344
```

```
{
  "name": "Nic Raboy",
  "email": "nic@couchbase.com",
  "address": "450 Mission Street",
  "city": "San Francisco",
  "state": "CA"
}
```

```
key: facebook::1234
```

```
{
  "profile": "f8d8d616-6ce0-4344"
}
```

```
key: twitter::1234
```

```
{
  "profile": "f8d8d616-6ce0-4344"
}
```



Relational Databases

Customers		
customer	firstname	lastname
2342	Nic	Raboy
Products		
product	name	
7182	Nintendo Switch	
Orders		
order	total	customer
1234	\$299.99	2342
Invoice		
order	product	quantity
1234	7182	2

```
SELECT
    c.firstname, c.lastname, p.name, o.total
FROM invoice AS i
JOIN order AS o ON i.order = o.order
JOIN product AS p ON i.product = p.product
JOIN customer AS c ON o.customer = c.customer
```



The Embedded Approach

Customers

customer	firstname	lastname
2342	Nic	Raboy

Products

product	name
7182	Nintendo Switch

Orders

order	total	customer
1234	\$299.99	2342

Invoice

order	product	quantity
1234	7182	2

```
{  
  "order": 1234,  
  "customer": {  
    "customer": "2342",  
    "firstname": "Nic",  
    "lastname": "Raboy"  
  },  
  "products": [  
    {  
      "product": 7182,  
      "name": "Nintendo Switch"  
    }  
  ],  
  "total": "$299.99"  
}
```



The Referenced Approach

Customers

Products

Orders

Invoice

{

 "firstname": "Nic",
 "lastname": "Raboy"

}

{

"name": "Nintendo Switch"

}

{

 "customer": 2342
 "total": "\$299.99"

}

{

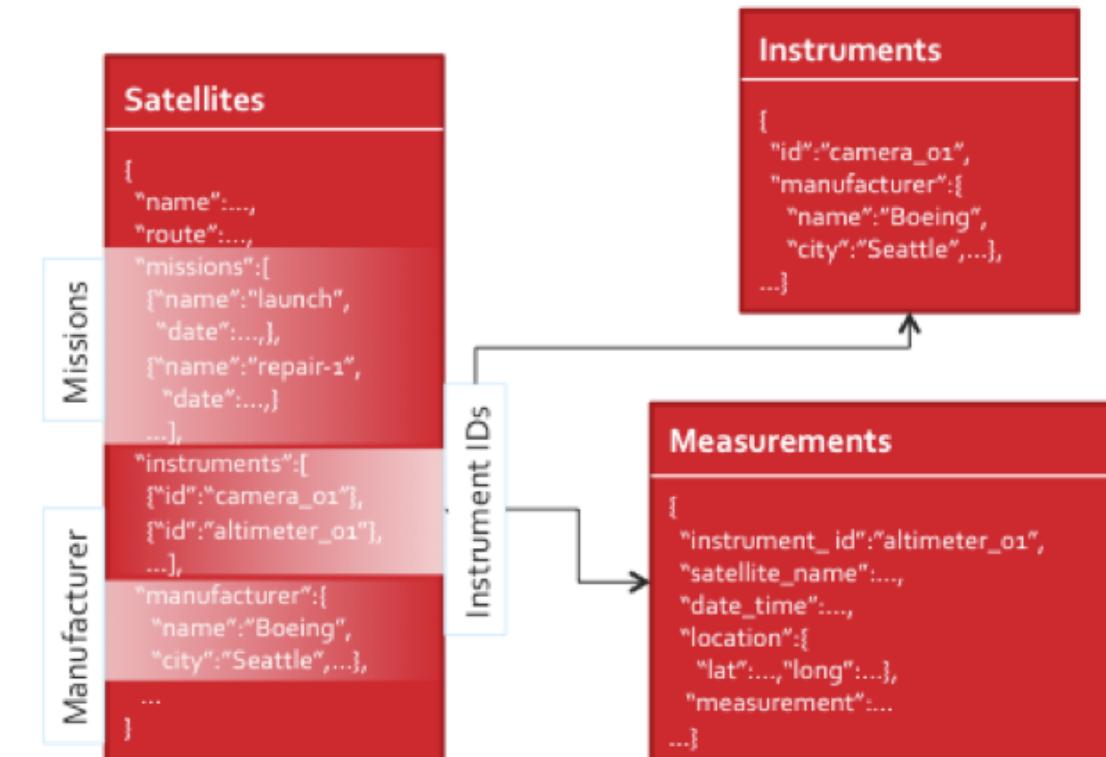
 "order": 1234,
 "product": 7182,
 "quantity": 2

}



Embedding vs Referencing

- Embed When
 - Both docs **are** frequently access together.
 - You need strict consistency between both docs.
- Reference When
 - Both docs **aren't** frequently access together.
 - You need strict consistency for the referenced information
 - Need to optimize for performance and referenced doc is large





Evolving Data Models

Change your code and you are good to go!

- Schema is App Driven in Couchbase Server
 - OK to maintain multiple versions of the schema
- Schema Changes with Couchbase Server
 - No Explicit Server Action Required by DBA
 - No Downtime required to update existing data to new shape



TRAVEL-SAMPLE DATA MODEL





Lab 2: Browse documents from the UI Console

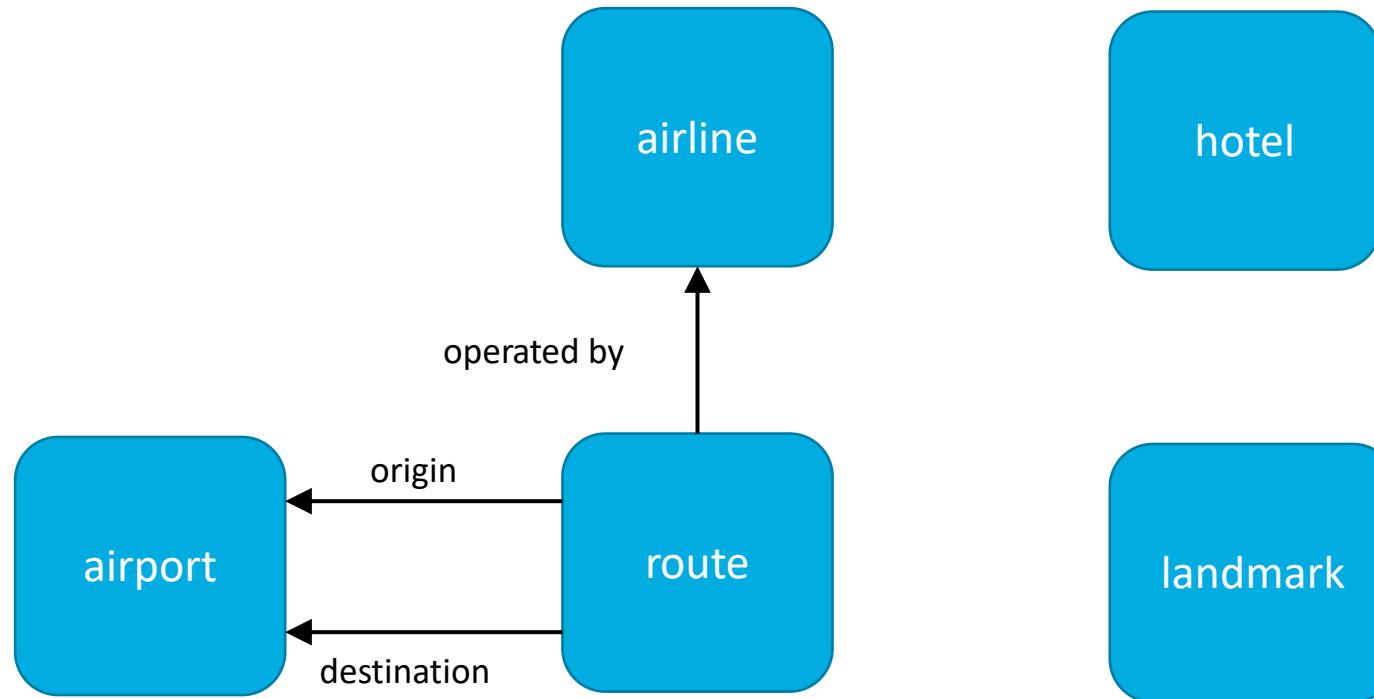
- Console > Buckets > travel-sample > Documents
- Try filter > start with > r
- Try Lookup ID: **route_10003**
- Create document. Add Document >
 - Document ID: k_1
 - Value {"name": "luis"}
- Delete document. ID: k_1

ID	Content	Actions
airline_10	{"callsign":"MILE-AIR","country":"United States","iata":"Q5","icao":"MLA","id":10,"name":"40-Mile Air","type":"airline"}	<button>Delete</button> <button>Edit</button>
airline_10123	{"callsign":"TXW","country":"United States","iata":"TQ","icao":"TXW","id":10123,"name":"Texas Wings","type":"airline"}	<button>Delete</button> <button>Edit</button>
airline_10226	{"callsign":"atifly","country":"United States","iata":"A1","icao":"A1F","id":10226,"name":"Atifly","type":"airline"}	<button>Delete</button> <button>Edit</button>
airline_10642	{"callsign":null,"country":"United Kingdom","iata":null,"icao":"JRB","id":10642,"name":"jc royal.britannica","type":"airline"}	<button>Delete</button> <button>Edit</button>
airline_10748	{"callsign":"LOCAIR","country":"United States","iata":"ZQ","icao":"LOC","id":10748,"name":"Locair","type":"airline"}	<button>Delete</button> <button>Edit</button>
airline_10765	{"callsign":"SASQUATCH","country":"United States","iata":"K5","icao":"SQH","id":10765,"name":"SeaPort Airlines","type":"airline"}	<button>Delete</button> <button>Edit</button>

Lab



Travel-sample data model





Travel-sample data model: airline

airline

- Sample key: airline_10123

```
{  
    "callsign": "TXW",  
    "country": "United States",  
    "iata": "TQ",  
    "icao": "TXW",  
    "id": 10123,  
    "name": "Texas Wings",  
    "type": "airline"  
}
```



Travel-sample data model: airport

- Sample key: airport_1256

airport

```
{  
    "airportname": "Les Loges",  
    "city": "Nangis",  
    "country": "France",  
    "faa": null,  
    "geo": {  
        "alt": 428,  
        "lat": 48.596219,  
        "lon": 3.006786},  
    "icao": "LFAI",  
    "id": 1256,  
    "type": "airport",  
    "tz": "Europe/Paris"  
}
```



Travel-sample data model: route

- Sample key: route_10004
- route

```
{  
  "airline": "AF",  
  "airlineid": "airline_137",  
  "destinationairport": "AMS",  
  "distance": 9442.50092891188,  
  "equipment": "777",  
  "id": 10004,  
  "sourceairport": "TPE",  
  "stops": 0,  
  "type": "route",  
}
```

```
"schedule": [  
  {  
    "day": 0,  
    "flight": "AF545",  
    "utc": "13:22:00"  
  },  
  {  
    "day": 0,  
    "flight": "AF350",  
    "utc": "01:21:00"  
  },  
  . . . ,  
  {  
    "day": 6,  
    "flight": "AF250",  
    "utc": "02:32:00"  
  } ]  
}
```



Travel-sample data model: hotel

- Sample key: hotel_10063

```
{  
    "address": "6 rue aux Juifs",  
    "alias": "Les Rouges Gorges",  
    "checkin": null,  
    "checkout": null,  
    "city": "Giverny",  
    "country": "France",  
    "description": "The rustic style ...",  
    "directions": "27620 Giverny",  
    "email": null,  
    "fax": null,  
    "free_breakfast": false,  
    "free_internet": true,  
    "free_parking": true,  
    "geo": {  
        "accuracy": "APPROXIMATE",  
        "lat": 49.078069,  
        "lon": 1.520866  
    },  
}
```

hotel

```
"id": 10063,  
"name": "The Robins",  
"pets_ok": false,  
"phone": null,  
"price": "60 / 70 euros",  
"public_likes": [],  
"reviews": [ ... ],  
"state": "Haute-Normandie",  
"title": "Giverny",  
"tollfree": null,  
"type": "hotel",  
"url":  
"http://givernyguesthouse.com/robin.htm",  
"vacancy": true  
}
```



Travel-sample data model: landmark

landmark

- Sample key: `landmark_10023`

```
{  
    "activity": "eat",  
    "address": "3 King Street, ME6 1EY",  
    "alt": null,  
    "city": "Gillingham",  
    "content": "Chinese restaurant just off the High Street.",  
    "country": "United Kingdom",  
    "directions": null,  
    "email": null,  
    "geo": {  
        "accuracy": "RANGE_INTERPOLATED",  
        "lat": 51.38697,  
        "lon": 0.54852  },  
    "hours": null,  
    "id": 10023,  
    "image": null,  
    "name": "Beijing Inn",  
    "phone": null,  
    "price": null,  
    "state": null,  
    "title": "Gillingham (Kent)",  
    "tollfree": null,  
    "type": "landmark",  
    "url": http://beijinginn.co.uk/div/  
}
```



JAVA SDK CONNECTING TO COUCHBASE





Including the SDK

```
// Gradle  
dependencies {  
    ...  
    compile group: 'com.couchbase.client', name: 'java-client', version:'2.5.4'  
    ...  
}
```

```
// Maven  
<dependencies>  
    <dependency>  
        <groupId>com.couchbase.client</groupId>  
        <artifactId>java-client</artifactId>  
        <version>2.5.4</version>  
    </dependency>  
</dependencies>
```

TIP. Check last version available here:

<https://developer.couchbase.com/documentation/server/current/sdk/java/start-using-sdk.html>



Connection Basics

```
// Java
import com.couchbase.client.java.CouchbaseCluster;
import com.couchbase.client.java.CouchbaseBucket;

CouchbaseCluster cluster = CouchbaseCluster.create("couchbase://<host>");
cluster.authenticate("<username>", "<password>");
CouchbaseBucket bucket = cluster.openBucket("<bucket-name>");
```

- Not like a JDBC Pool
- Reuse Bucket object
- Singleton pattern



Lab 3: Couchbase Connection

- Use source file: MainLab.java
- Implement method initConnection()
- Read config values from System properties:
 - cbworkshop.clusteraddress
 - cbworkshop.user
 - cbworkshop.password
 - cbworkshop.bucket
- Initialize bucket
- Run application
- Check output:

```
Nov 20, 2017 1:44:25 PM com.couchbase.client.core.CouchbaseCore <init>
INFO: CouchbaseEnvironment: {sslEnabled=false, sslKeystoreFile='null',
sslTruststoreFile='null', sslKeystorePassword=false, sslTruststorePassword=false,
sslKeystore=null, sslTruststore=null, bootstrapHttpEnabled=true,
bootstrapCarrierEnabled=true, bootstrapHttpDirectPort=8091, bootstrapHttpSslPort=18091,
... }
```

```
Nov 20, 2017 1:44:33 PM com.couchbase.client.core.config.DefaultConfigurationProvider$8
call
INFO: Opened bucket travel-sample
```

```
Nov 20, 2017 1:44:35 PM com.couchbase.client.core.node.CouchbaseNode signalConnected
INFO: Connected to Node 127.0.0.1/localhost
```

Lab



JAVA SDK KEY VALUE & CRUD





Creating Documents

- Data can be flat or complex
- Document keys can be custom, automatically generated, or incrementing
- The `insert` operator will create new documents if the key does not already exist
- The `upsert` operator will create or replace

```
JsonObject data = JsonObject.create()
    .put("firstname", "Nic")
    .put("lastname", "Raboy");
JsonArray address = JsonArray.create()
    .add(JsonObject.create().put("city", "Mountain View").put("state", "CA"))
    .add(JsonObject.create().put("city", "San Francisco").put("state", "CA"));
data.put("address", address);
bucket.insert(JsonDocument.create("person-1", data));
```



Retrieving Documents by Key

- Data can be retrieved using a key-value lookup or with a N1QL query
- Lookups are significantly faster than indexed queries with N1QL
- Java has can be asynchronous (non-blocking) with RxJava or synchronous

```
// Java  
bucket.get("person-1").content();
```



Lab 4: Create Object

- Implement method: `create(String[] words)`
- Compose a JSON document like this:
- Read parameters from command line:
 - Document key
 - from
 - to
- Compose key with prefix “msg::” + user provided
- Set timestamp to CurrentTimeMillis
- Set type to “msg”
- Use insert
- Try several times. See results in console
- Try same key (Error should appear!)
- Repeat with upsert

Key:

msg::some_text

```
{  
    "timestamp": 1511184840248,  
    "from": "luis",  
    "to": "daniel",  
    "type": "msg"  
}
```

Lab



Lab 5: Read Object

- Implement method: `read(String[] words)`
- Read parameter from command line:
 - Document key
- Read the document
- Write Json to STDOUT
- Test with values:
 - `airline_10226`
 - `route_10009`
 - `hotel_10904`

```
# read airline_10226
{"country": "United States", "iata": "A1", "callsign": "atifly", "name": "Atifly", "icao": "A1F", "id": 10226, "type": "airline"}  
..
```

- **Extra Bonus:** implement code to output a friendly message when document is not found

Lab



Lab 6: Update Object

- Implement method: update(String[] words)
- Read parameter from command line:
 - Document key (prefix with “airline_” from code)
- Read the document
- Modify attribute “name”: set same value UPPERCASE
- Use replace to modify

```
# read airline_10642
{"country": "United Kingdom", "iata": null, "callsign": null, "name": "Jc royal.britannica", "icao": "JRB", "id": 10642, "type": "airline"}
# update 10642
# read airline_10642
{"country": "United Kingdom", "iata": null, "callsign": null, "name": "JC ROYAL.BRITANNICA", "icao": "JRB", "id": 10642, "type": "airline"}  
..
```

Lab



Lab 7: Delete Object

- Implement method: delete(String[] words)
- Read parameter from command line:
 - Document key (prefix with “msg::” from code)
- Delete document
- Tip: use create, then delete same key

```
# create 1001 luis ana
# read msg::1001
{"from":"luis","to":"ana","type":"msg","timestamp":1511192232720}
# delete 1001
# read msg::1001
java.lang.NullPointerException
#         at com.cbworkshop.MainLab.read(MainLab.java:95)
at com.cbworkshop.MainLab.process(MainLab.java:60)
at com.cbworkshop.MainLab.main(MainLab.java:30)
```

Lab



JAVA SDK WORKING WITH PARTS OF A DOCUMENT





The Plan – Working with Parts of a Document

- Get Parts of a NoSQL JSON Document
- Update JSON Properties in a Document
- Batch Subdocument Operations Together



Large Documents

```
key: nraboy
```

```
{
  "profile": {
    "firstname": "Nic",
    "lastname": "Raboy"
  },
  "data": [
    // 20MB of data
  ]
}
```



Get Part of a Document

```
// Java
DocumentFragment<Lookup> result = bucket
    .lookupIn("nraboy")
    .get("profile")
    .execute();
```



Update Part of a Document

```
// Java
SubdocOptionsBuilder builder = new SubdocOptionsBuilder();
builder.createParents(true);
bucket().mutateIn("nraboy").upsert("profile.firstname", "Nicolas",
builder).execute();
```



Chain Subdocument Operations

```
DocumentFragment<Mutation> result = bucket
    .mutateIn("nraboy")
    .replace("profile.firstname", "Nic")
    .insert("profile.gender", "Male", builder)
    .remove("data")
    .withDurability(PersistTo.MASTER, ReplicateTo.NONE)
    .execute();
```

```
DocumentFragment<Lookup> result = bucket
    .lookupIn("myKey")
    .get("sub.value")
    .exists("fruits")
    .execute();
```

```
String subValue = result.content("sub.value", String.class);
boolean fruitsExist = result.content("fruits", Boolean.class);
```



Lab 8: SubDocument API example

- Implement method: subdoc(String[] words)
- Read parameter from command line:
 - Document key (prefix with “msg::” from code)
- Using SubDocument API:
 - Change the actual value of the “from” attribute to “administrator”
 - Add a new attribute: “reviewed”, with value CurrentTimeMillis

```
# create 1005 juan santiago
# read msg::1005
{"from":"juan","to":"santiago","type":"msg","timestamp":1511196994278}
# subdoc 1005
# read msg::1005
{"reviewed":1511197006619,"from":"Administrator","to":"santiago","type":"msg","timestamp":1511196994278}
```

Lab



INTRODUCTION TO N1QL: SQL FOR JSON





The Plan – Querying for Documents with N1QL

- Understand the Power of N1QL
- Use Query Workbench and the Couchbase CLI
- Create Indexes for N1QL Queries
- Query for Data with N1QL
- Create Data with N1QL
- Remove Data with N1QL



What is N1QL?

- **Non-first (N1) Normal Form Query Language (QL)**
 - It is based on ANSI 92 SQL
 - Its query engine is optimized for modern, highly parallel multi-core execution
- SQL-like Query Language
 - Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data
- N1QL extends SQL to handle data that is:
 - **Nested:** Contains nested objects, arrays
 - **Heterogeneous:** Schema-optional, non-uniform
 - **Distributed:** Partitioned across a cluster



Power of SQL

Flexibility of
JSON



N1QL (EXPRESSIVE)

- Access to every part of JSON document
- Scalar & Aggregate functions
- Subqueries in the FROM clause
- Aggregation on arrays

Expressive, familiar, and feature-rich language for querying,
transforming, and manipulating JSON data



N1QL (FAMILIAR)

- SELECT * FROM bucket WHERE ...
- INSERT single & multiple documents
- UPDATE any part of JSON document & use complex filter
- DELETE
- MERGE two sets of documents using traditional MERGE statement
- EXPLAIN to understand the query plan
 - EXPLAIN SELECT * FROM bucket WHERE ...

Expressive, familiar, and feature-rich language for querying, transforming, and manipulating JSON data



N1QL (FEATURE-RICH)

- Access to every part of JSON document
- Functions (Date, Pattern, Array, Conditional, etc)
 - <https://developer.couchbase.com/documentation/server/current/n1ql/n1ql-language-reference/functions.html>
- JOIN, NEST, UNNEST
- Covering Index
- Prepared Statements
- USE KEYS, LIKE

Expressive, familiar, and feature-rich language for querying,
transforming, and manipulating JSON data



N1QL – Example 1

```
SELECT      customers.id,
            customers.NAME.lastname,
            customers.NAME.firstname
            SUM(orderline.amount)
FROM        orders UNNEST orders.lineitems AS orderline
            JOIN customers ON KEYS orders.custid
WHERE       customers.state = 'NY'
GROUP BY    customers.id,
            customers.NAME.lastname
HAVING     SUM(orderline.amount) > 10000
ORDER BY    SUM(orderline.amount) DESC
```



N1QL – Example 2

Which airlines has each airport flying from?

```
SELECT r.sourceairport, ARRAY_AGG(DISTINCT a.name) as airlines
FROM `travel-sample` r
JOIN `travel-sample` a ON KEYS r.airlineid
WHERE r.type="route"
GROUP BY r.sourceairport
```



Query Workbench and the Couchbase CLI

- Query Workbench
 - <http://localhost:8091/ui/index.html#!/query/workbench>
- Couchbase CLI (Mac)
 - /Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/cbq
- Couchbase CLI (Windows)
 - C:\Program Files\CouchbaseServer\bin\cbq.exe
- Couchbase CLI (Linux)
 - /opt/couchbase/bin/cbq



CRUD IN N1QL

C

```
INSERT INTO `default`  
(KEY, VALUE)  
VALUES ("person-1", {  
    "firstname": "Nic",  
    "lastname": "Raboy"  
} )
```

R

```
SELECT name FROM `travel-sample` WHERE type="airline" AND icao="JRB"
```

U

```
UPDATE `travel-sample` USE KEYS ["hotel_10138"]  
SET pets_ok=true, free_internet=true
```

D

```
DELETE FROM `default`  
WHERE META().id = 'person-1'
```



GSI Indexes

Activity Documentation Support Administrator

CB5 Docker > Indexes

Global Indexes ▾ Views

Dashboard	bucket ▾	node	index name	storage type	status	build progress
Servers	sample	127.0.0.1:8091	#primary	Memory Optimized GSI	ready	100%
Buckets	travel-sample	127.0.0.1:8091	def_airportname	Memory Optimized GSI	ready	100%
Indexes	<p>Definition CREATE INDEX `def_airportname` ON `travel-sample`(`airportname`) WITH { "defer_build":true }</p>					
Search	travel-sample	127.0.0.1:8091	def_city	Memory Optimized GSI	ready	100%
Query	travel-sample	127.0.0.1:8091	def_faa	Memory Optimized GSI	ready	100%
XDCR	travel-sample	127.0.0.1:8091	def_icao	Memory Optimized GSI	ready	100%
Security	travel-sample	127.0.0.1:8091	def_name_type	Memory Optimized GSI	ready	100%
Settings	travel-sample	127.0.0.1:8091	def_primary	Memory Optimized GSI	ready	100%
Logs						



Primary Index

- The primary index is the index on the document key on every document in a keyspace
- The primary index is used for:
 - Complete keyspace scan – equivalent of table scan
 - Query does not have any predicates (filters)
 - Query has predicate on document key
 - No other index or access path can be used
- Primary index is optional

```
CREATE PRIMARY INDEX ON `travel-sample`;  
  
SELECT *  
FROM `travel-sample`;  
  
SELECT *  
FROM `travel-sample`  
WHERE META().id LIKE "user%";
```

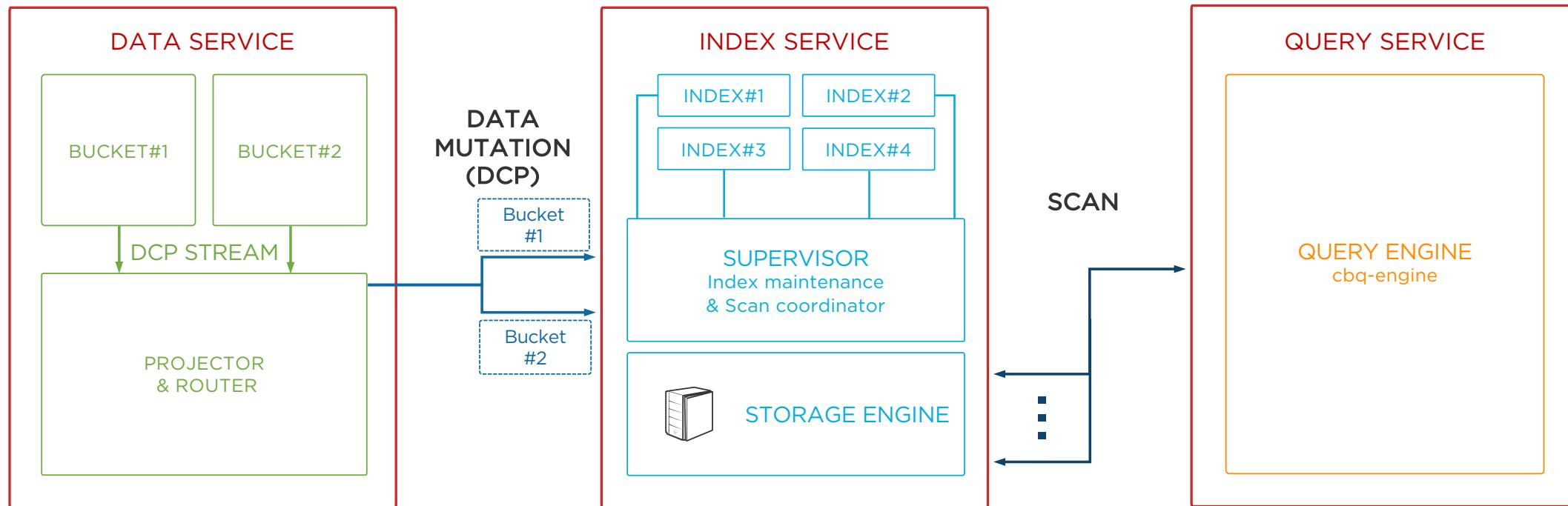


Secondary Index

- Simple Index
- Composite Index
- Functional Index
- Partial Index
- Replica Index
- Covering Index



Couchbase Services





JAVA SDK N1QL QUERIES





Query Raw String vs. DSL

Raw string query

```
N1qlQueryResult queryResult =  
bucket.query(N1qlQuery.simple("SELECT * FROM `travel-sample` LIMIT 10"));
```

Using the DSL

```
import static com.couchbase.client.java.query.Select.select;  
import static com.couchbase.client.java.query.dsl.Expression.*;  
  
...  
N1qlQueryResult query =  
bucket.query(select("*").from("`travel-sample`").limit(10));
```

Iterate result

```
for(N1qlQueryRow row : queryResult){  
    System.out.println(row.value().toString());  
}
```



Lab 9: Simple Query

- Implement method: query (String[] words)
- Execute the query: “SELECT * FROM `travel-sample` LIMIT 10”
- Print the results to STDOUT
- Use both implementations, raw and dsl

```
# query
{"travel-sample": {"country": "United States", "iata": "Q5", "callsign": "MILE-AIR", "name": "40-Mile Air", "icao": "MLA", "id": 10, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "TQ", "callsign": "TXW", "name": "Texas Wings", "icao": "TXW", "id": 10123, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "A1", "callsign": "atifly", "name": "Atifly", "icao": "A1F", "id": 10226, "type": "airline"}}
{"travel-sample": {"country": "United Kingdom", "iata": null, "callsign": null, "name": "JC ROYAL.BRITANNICA", "icao": "JRB", "id": 10642, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "ZQ", "callsign": "LOCAIR", "name": "Locair", "icao": "LOC", "id": 10748, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "K5", "callsign": "SASQUATCH", "name": "SeaPort Airlines", "icao": "SQH", "id": 10765, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "K0", "callsign": "ACE AIR", "name": "Alaska Central Express", "icao": "AER", "id": 109, "type": "airline"}}
 {"travel-sample": {"country": "United Kingdom", "iata": "5W", "callsign": "FLYSTAR", "name": "Astraeus", "icao": "AEU", "id": 112, "type": "airline"}}
 {"travel-sample": {"country": "France", "iata": "UU", "callsign": "REUNION", "name": "Air Austral", "icao": "REU", "id": 1191, "type": "airline"}}
 {"travel-sample": {"country": "France", "iata": "A5", "callsign": "AIRLINAIR", "name": "Airlinair", "icao": "RLA", "id": 1203, "type": "airline"}}
```

Lab



Query with parameters

Sample code

```
String sourceairport = ...;
String destinationairport = ...;

String queryStr = "SELECT a.name FROM `travel-sample` r JOIN `travel-sample` a ON
KEYS r.airlineid WHERE r.type=\"route\" AND r.sourceairport=$src AND
r.destinationairport=$dst";

JsonObject params = JsonObject.create()
    .put("src", sourceairport)
    .put("dst", destinationairport);

N1qlQuery query = N1qlQuery.parameterized(queryStr, params);
```



Lab 10: Query with parameters

- Implement method: queryAirports(String[] words)
- Read parameters from command line:
 - sourceairport
 - destinationairport
- Write a query to find airlines (airline names) flying from sourceairport to destinationairport.
Use JOIN
- Use a parametrized query
- **TIP:** Highest traffic airport codes: ATL, ORD, LHR, CDG, LAX, DFW, JFK

```
# queryairports JFK LHR
{"name": "British Airways"}
 {"name": "Delta Air Lines"}
 {"name": "American Airlines"}
 {"name": "US Airways"}
 {"name": "Virgin Atlantic Airways"}
 {"name": "Air France"}
```

Lab



Query Consistency

- `not_bounded`
 - Fastest
 - Returns data that is currently indexed and accessible by the index or the view.
- `at_plus`
 - Fast
 - A query submitted with `at_plus` consistency level requires all mutations, up to the moment of the `scan_vector` (the logical timestamp passed in with `at_plus`), to be processed before the query execution can start.
- `request_plus`
 - Slowest, but still fast
 - Requires all mutations, up to the moment of the query request, to be processed before the query execution can start.



Query Consistency

```
// Java
N1qlQueryResult result =
    bucket.query(
        N1qlQuery.simple(
            statement,
            N1qlParams.build().consistency(ScanConsistency.REQUEST_PLUS)
        )
    );
```



JAVA SDK BATCHING WITH RXJAVA





Batching with RxJava

- Implicit batching is performed by utilizing a few operators:
 - **Observable.just()** or **Observable.from()** to generate an Observable that contains the data you want to batch on.
- **flatMap()** to send those events against the Couchbase Java SDK and merge the results asynchronously.
- **last()** if you want to wait until the last element of the batch is received.
- **toList()** if you care about the responses and want to aggregate them easily.



Batching with RxJava

- The following example creates an observable stream of 5 keys to load in a batch,
- asynchronously fires off get() requests against the SDK,
- waits until the last result has arrived,
- and then converts the result into a list and blocks at the very end

```
Cluster cluster = CouchbaseCluster.create();
Bucket bucket = cluster.openBucket();

List<JsonDocument> foundDocs = Observable
    .just("key1", "key2", "key3", "key4", "key5")
    .flatMap(new Func1<String, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(String id) {
            return bucket.async().get(id);
        }
    })
    .toList()
    .toBlocking()
    .single();
```



Batching with RxJava

- If you wrap the code in a helper method, you can provide very nice encapsulated batching semantics

```
public List<JsonDocument> bulkGet(final Collection<String> ids) {  
    return Observable  
        .from(ids)  
        .flatMap(new Func1<String, Observable<JsonDocument>>() {  
            @Override  
            public Observable<JsonDocument> call(String id) {  
                return bucket.async().get(id);  
            }  
        })  
        .toList()  
        .toBlocking()  
        .single();  
}
```



Batching with RxJava - Batching mutations

- The following code generates a number of fake documents and inserts them in one batch.
- Note that you can decide to either collect the results with **toList()** as shown before or just use **last()** as shown here to wait until the last document is properly inserted.

```
// Generate a number of dummy JSON documents
int docsToCreate = 100;
List<JsonDocument> documents = new ArrayList<JsonDocument>();
for (int i = 0; i < docsToCreate; i++) {
    JsonObject content = JsonObject.create()
        .put("counter", i)
        .put("name", "Foo Bar");
    documents.add(JsonDocument.create("doc-"+i, content));
}

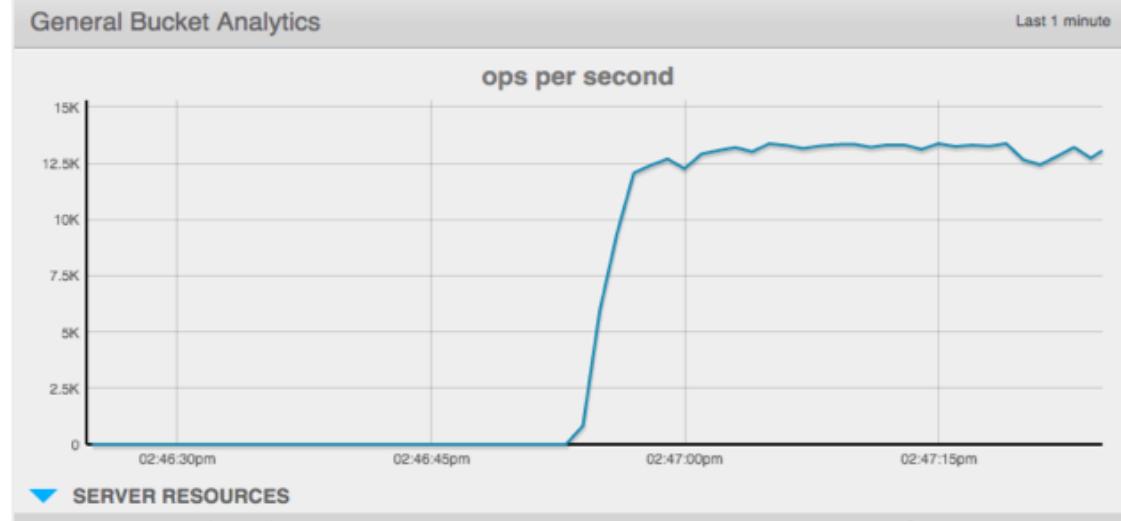
// Insert them in one batch, waiting until the last one is done.
Observable
    .from(documents)
    .flatMap(new Func1<JsonDocument, Observable<JsonDocument>>() {
        @Override
        public Observable<JsonDocument> call(final JsonDocument docToInsert) {
            return bucket.async().insert(docToInsert);
        }
    })
    .last()
    .toBlocking()
    .single();
```



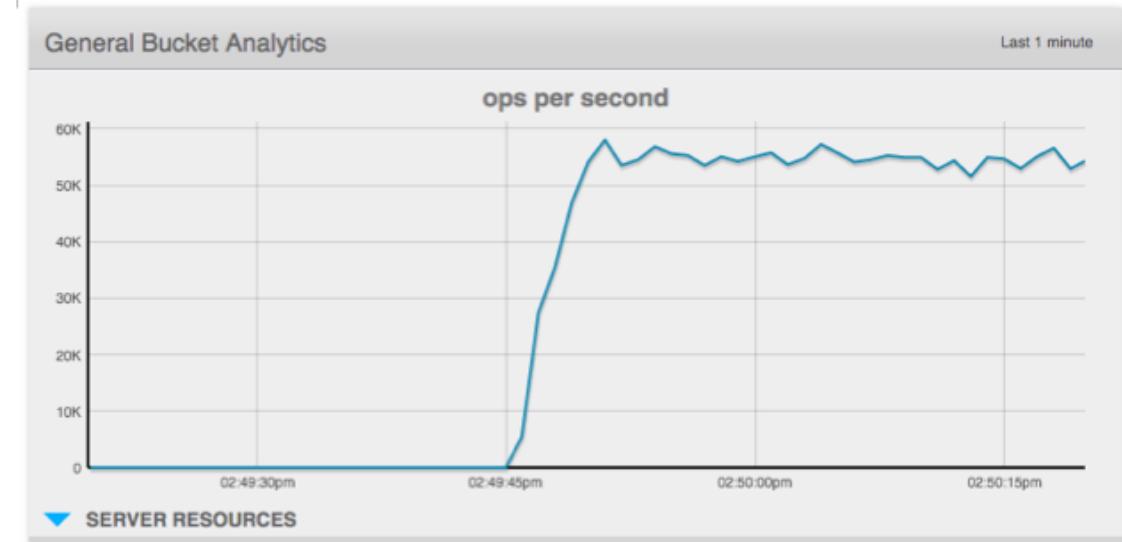
Sync Vs. ASync

- ✓ Here are two code samples, both synchronous, that showcase serialized and batched loading of documents.

```
// Serialized workload of loading documents
while(true) {
    List<JsonDocument> loaded = new ArrayList<JsonDocument>();
    int docsToLoad = 10;
    for (int i = 0; i < docsToLoad; i++) {
        JsonDocument doc = bucket.get("doc-" + i);
        if (doc != null) {
            loaded.add(doc);
        }
    }
}
```



```
// Same workload, utilizing batching effects
while(true) {
    int docsToLoad = 10;
    Observable
        .range(0, docsToLoad)
        .flatMap(new Func1<Integer, Observable<JsonDocument>>() {
            @Override
            public Observable<JsonDocument> call(Integer i) {
                return bucket.async().get("doc-"+i);
            }
        })
        .toList()
        .toBlocking()
        .single();
}
```





Bulk Read Async

```
List<String> priceKeys;  
List<JsonObject> res = Observable  
    .from(priceKeys)  
    .flatMap(k -> bucket.async().get(k))  
    .map(doc -> doc.content())  
    .toList()  
    .toBlocking()  
    .single();
```



Bulk Write Async

Async version:

```
List<JsonDocument> docs = ...;  
Observable  
    .from(docs)  
    .flatMap(doc -> bucket.async().insert(doc))  
    .last()  
    .toBlocking()  
    .single();
```

Sync version:

```
for(JsonDocument doc : docs) {  
    bucket.insert(doc);  
}
```



Lab 11: Bulk Write Performance

- Implement method: bulkWrite (String[] words) : Async version
- Implement method: bulkWriteSync (String[] words) : Sync version
- Read parameters from command line:
 - size: Number of messages to insert. Keys will be from msg::1 to msg::[size]
- Delete all messages in the bucket: DELETE FROM `travel-sample` WHERE type="msg"
- Create an array of JsonDocuments of messages
- Insert the messages into the bucket (in async / sync way)
- Print the time elapsed to STDOUT
- Compare results sync vs. async. Check both time and operations per second in the console

Async



Sync



Lab



Query Async mode

```
bucket.async()
    .query(select("*").from(i(`travel-sample`)).limit(5))
    .flatMap(result ->
        result.errors()
        .flatMap(e ->
            Observable.<AsyncN1qlQueryRow>error(new CouchbaseException("N1QL
Error/Warning: " + e)))
        .switchIfEmpty(result.rows())
    )
    .map(AsyncN1qlQueryRow::value)
    .subscribe(
        rowContent -> System.out.println(rowContent),
        runtimeError -> runtimeError.printStackTrace()
    );
}
```



Lab 12: Simple Query – Async version

- Implement method: queryAsync (String[] words)
- Execute the query: “SELECT * FROM `travel-sample` LIMIT 5”
- Print the results to STDOUT
- Use asynchronous implementation

```
# queryasync
# {"travel-sample": {"country": "United States", "iata": "Q5", "callsign": "MILE-AIR", "name": "40-Mile Air", "icao": "MLA", "id": 10, "type": "airline"}}
{"travel-sample": {"country": "United States", "iata": "TQ", "callsign": "TXW", "name": "Texas Wings", "icao": "TXW", "id": 10123, "type": "airline"}}
 {"travel-sample": {"country": "United States", "iata": "A1", "callsign": "atifly", "name": "Atifly", "icao": "A1F", "id": 10226, "type": "airline"}}
 {"travel-sample": {"country": "United Kingdom", "iata": null, "callsign": null, "name": "JC ROYAL.BRITANNICA", "icao": "JRB", "id": 10642, "type": "airline"}}
 {"travel-sample": {"country": "United States", "iata": "ZQ", "callsign": "LOCAIR", "name": "Locair", "icao": "LOC", "id": 10748, "type": "airline"}}
```

Lab



JAVA SDK FULL TEXT SEARCH





Full Text Search vs N1QL

- Wildcard Queries are Slow
- What happens to variations of the wildcard term in N1QL / SQL?
 - '%golang%', does 'Golang' or 'GoLang' pop up?
 - '%golang%', do spelling errors like 'goang' pop up?
- FTS is for Natural Language Querying



Lab 13: FTS – Create index

- Build an FTS Search index on travel-sample for hotels (type=“hotel”)
- Index only the attribute “description”
- Index name: sidx_hotel_desc
- Bucket: travel-sample
- Type Mappings: uncheck default
- Add type mapping: description
- Check “only index specified fields”
- Insert child field: description
- Search by term: farm

CB5 Docker > Full Text Search > Add Index

Dashboard Servers Buckets Indexes Search Query XDCR Security Settings Logs

Name: sidx_hotel_desc Bucket: travel-sample

Type Identifier:

JSON type field: type

Doc ID up to separator: delimiter

Doc ID with regex: regular expression

+ Add Type Mapping

Type Mappings

hotel | only index specified fields

field: description type: text searchable as: description analyzer: inherit

index store include in _all field include term vectors

default | disabled | dynamic

ok cancel

CB5 Docker > Full Text Search > Search Results

Search Results for sidx_hotel_desc

1. hotel_40177
2. hotel_15134
3. hotel_27626
4. hotel_1507
5. hotel_4399
6. hotel_28259
7. hotel_5846

Analyzers
Custom Filters
Advanced
Index Replicas 0

Create Index Cancel

Lab



Full Text Search: Java example

```
String term = ... ;  
  
MatchQuery fts = SearchQuery.match(term);  
SearchQueryResult result = bucket.query(new SearchQuery("sidx_hotel_desc", fts));  
for (SearchQueryRow row : result) {  
    System.out.println(row);  
}
```



Lab 14: FTS – Java example

- Implement method: search(String[] words) : Async version
- Read parameters from command line:
 - term: String to look for in the index
- Write code to search using the index `sidx_hotel_desc`
- Print results to STDOUT

```
# search pool
DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_18572d87', id='hotel_20420', score=5.412489826449426, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_20420", "score": 5.412489826449426}], offset=0}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_18572d87', id='hotel_40383', score=3.8272084673560847, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_40383", "score": 3.8272084673560847}], offset=1}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_18572d87', id='hotel_4396', score=3.8272084673560847, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_4396", "score": 3.8272084673560847}], offset=2}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_13aa53f3', id='hotel_3785', score=3.209767754362897, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_3785", "score": 3.209767754362897}], offset=3}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_6ddbfb54', id='hotel_25303', score=3.1616432709856555, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_25303", "score": 3.1616432709856555}], offset=4}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_f4e0a48a', id='hotel_3786', score=3.124082787542814, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_3786", "score": 3.124082787542814}], offset=5}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_6ddbfb54', id='hotel_20423', score=2.738063243874787, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_20423", "score": 2.738063243874787}], offset=6}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_18572d87', id='hotel_3714', score=2.6000760548180764, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_3714", "score": 2.6000760548180764}], offset=7}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_54820232', id='hotel_16631', score=2.440399959929308, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_16631", "score": 2.440399959929308}], offset=8}, DefaultSearchQueryRow{index='sidx_hotel_desc_13e37e3ca69ee59e_f4e0a48a', id='hotel_25671', score=2.386057638178849, explanation={}, locations=DefaultHitLocations{size=1, location=[{"id": "hotel_25671", "score": 2.386057638178849}], offset=9}
```

Lab



Types of Full Text Search Queries

- Match Query
 - A match query analyzes the input text and uses that analyzed text to query the index. An attempt is made to use the same analyzer that was used when the field was indexed.
- Match Phrase Query
- Fuzzy Query
- Wildcard Query
- Conjunction Query
- Range Queries
- <https://developer.couchbase.com/documentation/server/current/fts/fts-query-types.html>



JAVA SDK SPRING BOOT





Travel-sample demo on couchbaselabs

- This application uses Couchbase Server 5.0 together with Spring Boot, Angular2 and Bootstrap.
- Documentation: <https://github.com/couchbaselabs/try-cb-java/tree/5.0-updates>
- From command line execute:

```
git clone -b 5.0-updates https://github.com/couchbaselabs/try-cb-java.git
```

- To run the app execute:

```
cd try-cb-java/
```

```
mvn spring-boot:run -Dstorage.host=127.0.0.1 -Dstorage.bucket=travel-sample -Dstorage.password=password -Dstorage.username=travel
```

- Access application: <http://localhost:8080>
- Details: <https://blog.couchbase.com/traveling-with-couchbase-using-the-java-sdk/>



Travel-sample demo on couchbaselabs

The screenshot shows a web browser window titled "TryCb" at "localhost:8080/home". The user is logged in as "manuel". The navigation bar includes links to "HipChat", "ToRead", "Couchbase", "VECI", "Varios", and "GitTalent". A "Show Narration" button is also present. The main content area features the "CBTravel BETA" logo and a search form for flights from "John F Kennedy Intl" to "Los Angeles Intl" on travel dates "Leave 01/12/2018" and "Return 03/12/2018". A "Find Flights" button is located below the search fields. Below the search form, a section titled "Available Outbound Leg Flights" displays a table for the route "John F Kennedy Intl -> Los Angeles Intl". The table lists the following flights:

Airline	Flight	Departure	From	To	Aircraft	Price	Reservation
American Airlines	AA882	19:34:00	JFK	LAX	32B 762	561\$	Choose
American Airlines	AA380	11:49:00	JFK	LAX	32B 762	522\$	Choose
American Airlines	AA657	05:47:00	JFK	LAX	32B 762	1242\$	Choose
Delta Air Lines	DL320	22:16:00	JFK	LAX	76W 752	1803\$	Choose
Delta Air Lines	DL415	01:34:00	JFK	LAX	76W 752	934\$	Choose
Delta Air Lines	DL904	19:11:00	JFK	LAX	76W 752	966\$	Choose
Delta Air Lines	DL310	02:30:00	JFK	LAX	76W 752	66\$	Choose

THANK YOU

