



N1QL for SQL People

Indexing and Querying JSON Documents using N1QL

Couchbase 5.0+

Lab Workbook

Leo Schuman

August 8, 2017

Lab 1 - Install Couchbase, create bucket, load data, create primary index, and install tools

Objectives

A	Install and configure Couchbase Server 5.x as a single node cluster
B	Survey the Couchbase administration console
C	Create and configure a Couchbase bucket
D	Load documents into Couchbase using <i>cbimport</i> or <i>cbdocloader</i>
E	Create a primary index to support ad hoc queries (CREATE PRIMARY INDEX)

If you have an existing Couchbase installation you wish to restore later, archive the data and configuration files from these locations.

macOS	~/Library/Application Support/Couchbase/var/lib/couchbase
Windows	C:\Program Files\couchbase\server\var\lib\couchbase
Linux	/opt/couchbase/var/lib/couchbase

A. *Install and configure Couchbase Server 5.x as a single node cluster*

You want to run Couchbase from your local system for learning purposes.

1. Download *Couchbase Server Enterprise Edition 5.x* for your operating system.
<http://www.couchbase.com/downloads>
2. Review the Release Notes, and install as described in the documentation for your OS:
<http://docs.couchbase.com>
Note, on Windows, you must run the installer using elevated Administrator permissions.
To explore container-based installation, see:
<http://www.couchbase.com/containers>.
3. Unless the Couchbase Setup tool launches automatically, open a web browser, and browse to this URL to launch the Setup tool.
<http://localhost:8091>

4. Select *Setup New Cluster*



5. In the Setup tool, review all settings and accept all defaults, except for changing these:

- Cluster Name: *Training Cluster*
- Admin Username: *Administrator*
- Password: *password*
- Accept the Terms and Conditions (*check*)
- Configure Disk, Memory, Services (*click*)
 - Data Service (*check*): *2048mb*
 - Index (GSI) Service (*check*): *1024mb*
 - Search (FTS) Service (*check*): *512mb*
 - Query Service (*check*)
 - Memory-Optimized Global Secondary Indexes (*check*)

Note, if requested by your local firewall, accept incoming network connections for *beam.smp*, *memcached*, *epmd*, *indexer*, *moxi*, *projector*, *cbq-engine*, and *cbft*. A full list of port requirements is available here:

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

Note, the settings for this course run a low-impact one node Couchbase cluster on a local system for non-performance related learning purposes, only. Couchbase advises a minimum of 3 nodes for any production cluster. Please read the documentation.

6. After completing Setup, you should see this screen. If not, browse <http://localhost:8091>.

B. Survey the Couchbase administration console

You want a basic orientation to the Couchbase UI.

7. In the Couchbase UI, navigate to and briefly review each top-level screen.

For a solid introduction to the essential concepts of Couchbase technology, please take this free online training at <http://training.couchbase.com/online>.

CB030 - Essentials of Couchbase Engaged DBMS Technology

Documentation is available at <http://docs.couchbase.com>

The screenshot shows the Couchbase Dashboard interface. At the top, it says "Training Cluster > Dashboard". On the left, there's a sidebar with links: Dashboard (selected), Servers, Buckets (highlighted with a yellow star icon), Indexes, Search, Query, XDCR, Security, Settings, and Logs. The main area has two sections: "Data Service" (1 node) and "GSI Service" (1 node). Below that is a chart titled "Data Service Memory" showing memory usage: "Total Allocated (0 B)" with a bar divided into "In Use (0 B)" and "Unused (0 B)". Under "Buckets", it says "0 active" and "You have no data buckets."

C. Create and configure a Couchbase bucket

You want to create the basic Couchbase data container, a bucket.

8. In the Couchbase UI, select the *Buckets* panel, and choose *Add Bucket*.

The screenshot shows the Couchbase Buckets panel. At the top, it says "Training Cluster > Buckets". On the left, there's a sidebar with links: Dashboard, Servers, Buckets (highlighted with a yellow star icon), Indexes, and Search. The main area shows a table with columns: "name", "items", and "disk used". A red arrow points from the sidebar's "Buckets" link to the "name" column header. Another red arrow points from the "ADD BUCKET" button in the top right to the "name" column header. Below the table, it says "You have no data buckets." and "With data & indexes."

9. In the *Add Data Bucket* panel, set the following values.

- Name: *couchmusic2*
- Memory Quota: *1024mb*
- Advanced Settings (*click*)
 - Replicas: *Disable (uncheck)*
 - Flush: *Enable (check)*

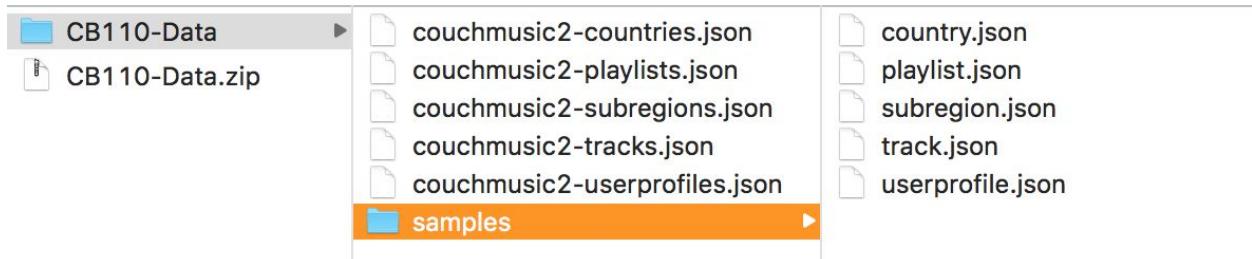
Note, because replicas are created on distinct nodes, replication requires a minimum of two nodes. So, for this single node training cluster, you will disable replication for this bucket. In production, Couchbase recommends a minimum of three nodes per cluster.

10. Click the bucket name to review summary information for the new *couchmusic2* bucket.

D. Load documents into Couchbase using cbimport

You want to bulk load data into the bucket you have created.

- From the Learning Management System, Download the *CB110-Data.zip* file and extract it to your *desktop* or similar directory. It contains JSON files of five different types of JSON document (object), related to an application named *couchmusic2*, along with a PDF document describing their structure.



- Open a Terminal (Command) window, navigate to the Couchbase *bin* folder, and briefly review the contents of this folder. Notice the *cbimport*, *cbq*, and *couchbase-cli* tools.

macOS	/Applications/Couchbase Server.app/Contents/Resources/couchbase-core/bin/
Windows	C:\Program Files\Couchbase\Server\bin\
Linux	/opt/couchbase/bin/

```
[couchbase:bin Schuman$ ls
c_rehash
cbbackup
cbbackupmgr
cbbackupwrapper
cbbrowse_logs
cbcollect_info
cbcompact
cbdocloader
cbdump-config
cbenable_core_dumps.sh
cbeptl
cbexport
cbft
chft-bleve
cbimport
cbindex
cbindexperf
cbindexplan
cbq
cbq-engine
cbq.old
cbrecovery
cbrestore
cbvbucketctl
cbvdiff
cbworkloadgen
couch_compact
couch_dbck
couch_ddump
couch_dbinfo
couch_view_file_merger
couch_view_group_cleanup
couch_view_group_comparator
couch_view_index_builder
couch_view_index_updater
couchbase-cli
couchbase-server
couchdb
couchjs
couchjs.tpl
ct_run
dbdiff
dump-guts
dump-stats
epmd
erl
gomeata
goport
gosecrets
goxdcr
gozip
indexer
install
jeprof
kv_trace_dump
mcbp_packet_printer
mctl
mlogsplit
mcstat
mctimings
memcached
mossScope
moxi
openssl
plasma_dump
priv
projector
saslauthd-port
sigar_port]
```

Note, *cbimport* is a robust data loading tool released as a new feature of Couchbase 5.0. For full detail on using *cbimport*, see here:

<https://developer.couchbase.com/documentation/server/current/tools/cbimport.html>

- Add this *bin* folder to the PATH environment variable for your operating system, so that its commands may be invoked from any command line location. If needed, look up your operating system documentation for details on this process.

14. In the Terminal, navigate back to the `/CB110-Data` folder.
15. In a text editor, open `CB110-Data/couchmusic2-countries.json`, and examine its contents. Notice each line contains a JSON document (object).



```

couchmusic2-userprofiles.json
1 [{"status": "active", "picture": {"large": "https://randomuser.me/api/portraits/women/88.jpg", "medium": "https://randomuser.me/api/portrai
2 tive/88.jpg", "small": "https://randomuser.me/api/portraits/women/32.jpg"}, "name": {"title": "Miss", "first": "Aahine", "last": "Geffeten", "suffix": null}, "gender": "female", "dob": {"age": 35, "date": "1983-07-12"}, "location": {"city": "Paris", "country": "France", "lat": 48.8566, "lon": 2.3522}, "phone": "+33 1 5555 1234", "email": "aahinge@effeteness42037.com", "username": "aahinge@effeteness42037", "password": "636172626f6e", "favoriteGenres": ["Classical Crossover", "Contemporary Blues", "French Pop", "Progressive Bluegrass"], "lastName": "Riley", "title": "Mrs", "type": "userprofile", "email": "delores.riley@hotmail.com"}}
3
4
5
6
7

```

16. Open the sample of this document type, `CB110-Data/samples/userprofile.json`, and examine its structure. Notice its `type` and `username` properties.



```

userprofile.json
1 {
2   "status": "active",
3   "picture": {
4     "large": "https://randomuser.me/api/portraits/men/18.jpg",
5     "medium": "https://randomuser.me/api/portrai
6 tive/18.jpg",
7     "small": "https://randomuser.me/api/portraits/men/36.jpg"
8   },
9   "name": {
10     "title": "Mr",
11     "first": "Delores",
12     "last": "Riley",
13     "suffix": null
14   },
15   "gender": "male",
16   "dob": {
17     "age": 32,
18     "date": "1985-04-19"
19   },
20   "location": {
21     "city": "Paris",
22     "country": "France",
23     "lat": 48.8566,
24     "lon": 2.3522
25   },
26   "phone": "+33 1 5555 1234",
27   "email": "delores.riley@hotmail.com",
28   "username": "aahinge@effeteness42037",
29   "password": "636172626f6e",
30   "favoriteGenres": [
31     "Classical Crossover",
32     "Contemporary Blues",
33     "French Pop",
34     "Progressive Bluegrass"
35   ],
36   "lastName": "Riley",
37   "title": "Mrs",
38   "type": "userprofile",
39   "email": "delores.riley@hotmail.com"
}

```

17. Use `cbimport` to load `country` JSON documents to the `couchmusic2` bucket. Use a key pattern to generate a key for each document, as shown below.

- Cluster (-c): `couchbase://127.0.0.1`
- Username (-u): `Administrator`
- Password (-p): `password`
- Bucket (-b): `couchmusic2`
- Format (-f): `lines`
- Dataset (-d): `file://couchmusic2-userprofiles.json`
- Threads (-t): 2
- Key Pattern to Generate (-g): `%type%::%username%`

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines
-d file://couchmusic2-userprofiles.json -t 2 -g %type%::%username%
```

Note, if you are using Windows 10, you must escape % delimiters on the command line, using the ^ character, if the variable name referenced using those delimiters is also used as a system variable name, such as `username`. Modify the command show as:

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines
-d file://couchmusic2-userprofiles.json -t 2 -g %type%::^%username^%
```

```
[couchbase:CB110-Data Schuman$ ls
couchmusic2-countries.json      couchmusic2-subregions.json      samples
couchmusic2-data-model.pdf       couchmusic2-tracks.json
couchmusic2-playlists.json       couchmusic2-userprofiles.json
[couchbase:CB110-Data Schuman$ cbimport json -c couchbase://127.0.0.1 -u Administrator -p password ]
-b couchmusic2 -f lines -d file://couchmusic2-userprofiles.json -t 2 -g %type%::%username%
Json `file://couchmusic2-userprofiles.json` imported to `http://127.0.0.1:8091` successfully
couchbase:CB110-Data Schuman$ ]
```

18. In the Couchbase UI, open the *couchmusic2* bucket, examine the number of documents loaded, and open the Documents screen.

The screenshot shows the Couchbase UI Buckets screen. On the left, there's a sidebar with links: Dashboard, Servers, Buckets (which is highlighted with a red circle), and Indexes. The main area has a table with columns: name, items, resident, ops/sec, RAM used/quota, and disk used. A row for the 'couchmusic2' bucket is selected, showing 49,981 items, 100% resident, 0 ops/sec, 52.3MB / 1GB RAM used, and 37.7MB disk used. There are 'Documents' and 'Statistics' buttons at the bottom right of the row.

19. In the Documents screen, open the first document for editing.

The screenshot shows the Couchbase UI Documents screen. The sidebar shows 'Buckets' selected. The main area lists documents from the 'couchmusic2' bucket. The first document, 'userprofile::aahingeffeteness42037', is highlighted with a red box. It has an ID of 'content sample' and a JSON value: {"status": "active", "picture": {"large": "https://randomuser.me/api/portraits/women/88.jpg", "medium": "https://randomuser.me/api/portraits/med/women/88.jpg", "thumbnail": "https://randomuser.me/api/portraits/thumb/women/88.jpg"}. There are 'Delete' and 'Edit' buttons to the right of the document. Another document, 'userprofile::aahingheadwaiter24314', is partially visible below it.

20. Notice the metadata related to this document.

The screenshot shows the Couchbase UI Documents Editing screen for the document 'userprofile::aahingeffeteness42037'. The sidebar shows 'Logs' selected. The main area displays the document's JSON content. A red box highlights the 'id' field in the JSON, which is 'userprofile::aahingeffeteness42037'. A red arrow points from this highlighted 'id' to another red box containing the full JSON object: { "id": "userprofile::aahingeffeteness42037", "rev": "1-14d4af810e7a00000000000000000006", "expiration": 0, "flags": 33554438 }. There are 'Delete', 'Save As...', and 'Save' buttons at the top right.

21. Modify and run *cbimport* four more times as shown below, to lead each of the remaining document sets, using their type and the specified value from each document as a key.

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-playlists.json -t 2 -g %type%::%id%
```

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-subregions.json -t 2 -g %type%::%region-number%
```

```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-tracks.json -t 2 -g %type%::%id%
```

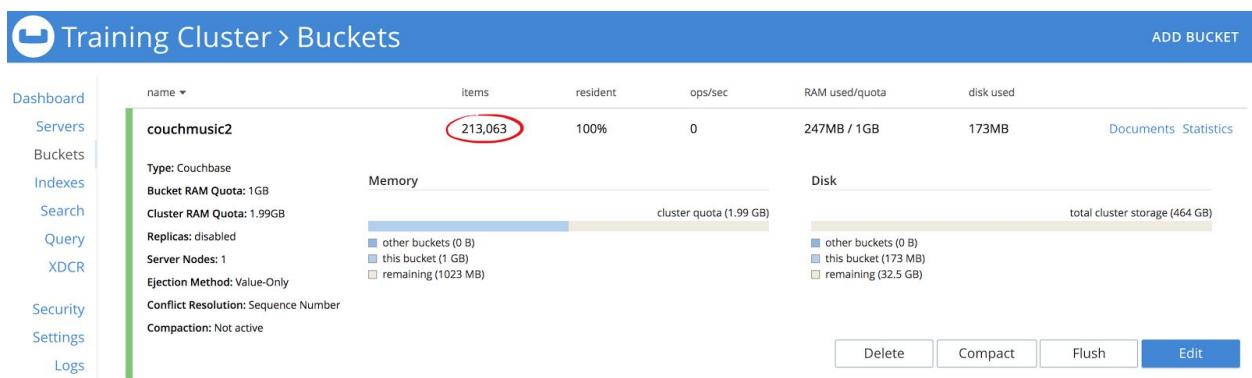
```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-countries.json -t 2 -g %type%::%countryCode%
```

Note, it is a common JSON design pattern to prefix document keys with a type identifier (e.g., *country*::*ES*). When using *cbimport*, prefixing can be specified using key generation, as shown above.

For an introduction to data modeling in JSON, including key patterns, please take this free Online Training course at <http://training.couchbase.com/online>.

CB105 - Introduction to Data Modeling in JSON

22. In the Couchbase UI, verify you've loaded 213,063 documents to a *couchmusic2* bucket, and open the *Documents* screen.



E. Create a primary index to support ad hoc queries (CREATE PRIMARY INDEX)

You want to enable ad hoc N1QL querying for the bucket you've created.

19. In the Couchbase UI, select the Query tab to open the Query Workbench.
20. To allow ad hoc N1QL queries to be run on the *couchmusic2* bucket, run a *CREATE PRIMARY INDEX* statement on this bucket, using the global indexing service.

```
CREATE PRIMARY INDEX ON `couchmusic2` USING GSI;
```

The screenshot shows the Couchbase Query Workbench. In the Query Editor, the command `CREATE PRIMARY INDEX ON `Customer360` USING GSI;` is entered and successfully executed. The Bucket Insights panel on the right indicates that the bucket `Customer360` is now `Fully Queryable`. A yellow star-shaped cursor points to the `Execute` button in the Query Editor.

Note, once a primary index is created on a bucket, it will appear as "Fully Queryable" in the *Bucket Insights* display. A primary index enables ad hoc queries for development and learning purposes.

In upcoming lessons, this course teaches how to create performant queries using secondary indexing. On production systems, you may choose to drop the primary index to improve system performance by eliminating the cost of maintaining it for new writes.

```
DROP PRIMARY INDEX ON [bucket name] USING GSI;
```

Do not drop the primary index on `couchmusic2`, as it will be relied upon in later Labs in this course, for learning purposes.

End of Lab

Lab 2 - Selecting documents and limiting results in *Query Workbench* and the *cbq* command line tool

Objectives

A	Use Query Workbench to select all attributes from all documents, save the results, and limit selection (*, SELECT, LIMIT)
B	Cancel a long-running query
C	(Optional) Execute a query using the <i>cbq</i> command line query tool

A. Use Query Workbench to select all attributes from all documents, and limit selection (*, SELECT, LIMIT)

1. In the Couchbase Server UI, select the *Query* tab to open the *Query Workbench*.
2. Write and execute a query to SELECT all attributes from all documents FROM *couchmusic2*, but LIMIT your result to 10 documents.

```
SELECT *
FROM couchmusic2
LIMIT 10;
```

Notice the *Status*, *Elapsed*, *Execution*, *Result Count*, and *Result Size* metrics displayed over the result, and review the other display and storage capabilities of the UI.

The screenshot shows the Couchbase Query Workbench interface. At the top, there are tabs for "Query Workbench" and "Query Monitor". Below the tabs, the "Query Editor" section contains a red box around the following SQL query:

```
1 SELECT *
2 FROM couchmusic2
3 LIMIT 10;
```

Below the editor, there are buttons for "Execute" (with a yellow star icon), "Explain", and a status message "success | elapsed: 10.83ms | execution: 10.81ms | count: 10 | size: 3177". To the right of the status message is a "Preferences" link. The "Query Results" section displays the JSON output of the query, which includes the following document:1 [{ "couchmusic2": { "countryCode": "AD", "gdp": 40214, "name": "Andorra", "population": 80792, "region-number": 39, "type": "country", "updated": "2015-10-01T07:35:13" } }, { }

At the bottom of the results section are buttons for "JSON", "Table", "Tree", "Plan", and "Plan Text".

3. Toggle between different displays for the results.

Query Results

countryCode	gdp	name	population	region-number	type	updated
AD	40214	Andorra	80792	39	country	2015-10-01T07:35:13
AE	41433	United Arab Emirates	9693829	145	country	2015-09-11T08:31:38
AF	650	Afghanistan	32059823	34	country	2015-09-03T21:08:49
AG	13812	Antigua and Barbuda	91873	29	country	2015-09-12T19:47:41
AI	0	Anguilla	14640	29	country	2015-09-04T22:15:09
AL	4184	Albania	3195196	39	country	2015-09-12T20:19:13
AM	3640	Armenia	2989427	145	country	2015-09-01T20:26:21
AN	0	Netherlands Antilles	0	29	country	2015-09-27T05:45:24
AO	5750	Angola	22850336	17	country	2015-09-16T13:43:28
AP	0	Asia Pacific	0	990	country	2015-09-26T02:02:08

4. Open the query export tool.

The screenshot shows the 'Training Cluster > Query' interface. At the top right, there is a 'IMPORT' button with a red arrow pointing to it, and a 'EXPORT' button with a yellow star icon. Below the buttons, the 'Query Workbench' dropdown is open, showing options like 'Dashboard', 'Servers', 'Buckets', and 'Query Editor'. The 'Query Editor' tab is selected, displaying a SQL query:

```
1 SELECT *
2 FROM couchmusic2
3 LIMIT 10;
```

5. Export the query results to the desktop as a JSON file.

The screenshot shows the 'Export Query / Data' dialog box. It has a 'Choose Export' section with a radio button for 'Query Results' (which is selected) and a checkbox for 'Query Statement'. Below that is a 'Filename' input field containing 'select-10-from-couchmusic2.json'. At the bottom are 'Cancel' and 'Save' buttons.

6. Examine the query results in a text editor.

The screenshot shows a code editor window with the file 'select-10-from-couchmusic2.json' open. The content of the file is a JSON array of objects representing countries:

```
1 [
2   {
3     "couchmusic2": {
4       "countryCode": "AD",
5       "gdp": 40214,
6       "name": "Andorra",
7       "population": 80792,
8       "region-number": 39,
9       "type": "country",
10      "updated": "2015-10-01T07:35:13"
11    }
12  },
13  {
14    "couchmusic2": {
```

B. Cancel a long-running query

7. Modify the prior query by removing the LIMIT clause, then execute the query.

```
SELECT *
FROM couchmusic2;
```

Notice the *Execute* button changes to a *Cancel* button during long query execution.

The screenshot shows the Query Editor and Query Results sections of the Query Workbench. In the Query Editor, the following SQL statement is entered:

```
1 SELECT *
2 FROM couchmusic2;|
```

In the Query Results section, the status of the query is shown as "Executing". A red box highlights the "Cancel" button, which is currently active (blue). Another red box highlights the status message "Executing statement".

Query Editor

1 SELECT *
2 FROM couchmusic2;|

Cancel ⏹ Explain ⏹ Executing | elapsed: | execution: | count: | size: 0 Preferences

Query Results

1 {"status": "Executing statement"} JSON Table Tree Plan Plan Text

8. Cancel the running query.

B. (Optional) Execute a query using the cbq command line query tool

9. In a Terminal window, launch the *cbq* tool located in the Couchbase *bin* directory. Authenticate to your local database engine using the Administrator credentials you created above.

```
cbq -e localhost:8091 -u Administrator -p password
```

```
[couchbase:CB110-Data Schuman$ [couchbase:CB110-Data Schuman]$ cbq -e localhost:8091 -u Administrator -p password ] ] Connected to : http://localhost:8091/. Type Ctrl-D or \QUIT to exit.
```

Path to history file for the shell : /Users/Schuman/.cbq_history
cbq> ↙

10. Select all attributes from *couchmusic2*, limiting the results to 1 document, and compare the resulting output to that seen when using Query Workbench.

```
SELECT *
FROM couchmusic2
LIMIT 1;
```

```
[couchbase:CB110-Data Schuman$ 
[couchbase:CB110-Data Schuman$ cbq -e localhost:8091 -u Administrator -p password
Connected to : http://localhost:8091/. Type Ctrl-D or \QUIT to exit.

Path to history file for the shell : /Users/Schuman/.cbq_history
cbq> SELECT *
>   FROM couchmusic2
>   LIMIT 1;
{
  "requestID": "1cd43c73-2256-4ed1-88b3-2cd1d0378157",
  "signature": {
    "*": "*"
  },
  →"results": [
    {
      "couchmusic2": {
        "countryCode": "AD",
        "gdp": 40214,
        "name": "Andorra",
        "population": 80792,
        "region-number": 39,
        "type": "country",
        "updated": "2015-10-01T07:35:13"
      }
    }
  ],
  →"status": "success",
  →"metrics": {
    "elapsedTime": "6.6608ms",
    "executionTime": "6.64484ms",
    "resultCount": 1,
    "resultSize": 314
  }
}
cbq> █
```

10. Examine the help for cbq.

```
[couchbase:CB110-Data Schuman$ 
[couchbase:CB110-Data Schuman$ cbq -h
Usage of cbq:
-b string
    Shorthand for -batch (default "off")
-batch string
    Batch mode for sending queries to Asterix. Values : on/off (default "off")
-c string
    Shorthand for -credentials
-credentials string
    A list of credentials, in the form user:password.
    For example : -c beer-sample:pass
-e string
    Shorthand for -engine (default "http://localhost:8091/")
-engine string
    URL to the query service/cluster.
    http://localhost:8091
```

Note, this course is taught using Query Workbench in the Couchbase console, not cbq.

End of Lab

Lab 3 - Selecting nested attributes, aliasing, concatenating, and accessing documents by key

Objectives

A	Use keys to select specific documents (USE KEYS)
B	Select named and nested attributes
C	Use aliases, concatenate values, and access metadata (AS, , META())

A. Use keys to select specific documents (USE KEYS)

1. In the Couchbase console, open the *Documents* view for the *couchmusic2* bucket.

The screenshot shows the Couchbase Training Cluster interface. On the left, there's a sidebar with links: Dashboard, Servers, Buckets, Indexes, Search, and Query. The 'Buckets' link is highlighted. The main area is titled 'Training Cluster > Buckets'. It lists four buckets: 'beer-sample', 'couchmusic2', and 'travel-sample'. The 'couchmusic2' bucket is selected, indicated by a red arrow pointing to its row. Each bucket row contains columns for name, items, resident, ops/sec, RAM used/quota, disk used, and two links: 'Documents' and 'Statistics'. A yellow starburst icon is positioned next to the 'Documents' link for the selected bucket.

2. In the *Documents* view, filter the displayed documents to start with keys containing the word "userprofile".

The screenshot shows the 'Documents' view for the 'couchmusic2' bucket. At the top, there's a dropdown menu set to 'couchmusic2'. Below it is a search bar with the placeholder 'ID'. To the right of the search bar is a 'filter' input field containing the URL-encoded filter: '?startkey=%22userprofile%22&skip=0&include_docs=true&limit=11'. A yellow starburst icon is positioned above the filter input. Below the search bar, there are two input fields: 'startkey' and 'endkey'. The 'startkey' field has the value 'userprofile' and is highlighted with a red box. A red arrow points from the 'userprofile' text in the filter input to this field. There's also a checkbox labeled 'inclusive_end' which is unchecked. At the bottom right are 'Reset' and 'Save' buttons, with a yellow starburst icon above the 'Save' button.

- Select *Edit Document* to open the first *userprofile* document, and copy its key (aka, "id"), displayed in its metadata.

```

1 {
2   "status": "active",
3   "picture": {
4     "large": "https://randomuser.",
5     "medium": "https://randomuser",
6     "thumbnail": "https://randomu.
7   },
8   "gender": "female",
9   "firstName": "Delores",

```

```

1 {
2   "id": "userprofile::aahingeffeteness42037",
3   "rev": "1-
4   14d542737bd40000000000000000006",
5   "expiration": 0,
6   "flags": 33554438

```

- In *Query Workbench*, select all attributes of this document by its key. Note, you must assign the key as a quoted string.

```

SELECT *
FROM couchmusic2
USE KEYS "userprofile::aahingeffeteness42037";

```

Note, the response time to retrieve this single document is dramatically faster than prior queries, as no bucket scan is needed. A specific document is retrieved from memory.

Query Editor

```

1 SELECT *
2 FROM couchmusic2
3 USE KEYS "userprofile::aahingeffeteness42037";|

```

Execute Explain success | elapsed: 1.48ms | execution: 1.46ms | count: 1 | size: 1581 Preferences

Query Results

JSON Table Tree Plan Plan Text

```

1 [
2   {
3     "couchmusic2": {
4       "address": {
5         "city": "warren",
6         "countryCode": "US",
7         "postalCode": 63450,
8         "state": "oregon",
9         "street": "6174 elgin st"
10      },
11      ...
12    }
13  ]

```

Note, the response time from a low memory single node cluster on a local laptop or desktop system is slow relative to the likely configuration of a production grade cluster.

- Repeat the steps above to copy a second Document ID (aka, "key") from *couchmusic2*.

6. Select all attributes for both documents, by their keys, in a single query. Note, you must use inline array syntax (comma separated values within brackets) for the key list, with each key quoted as a string value.

```
SELECT *
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Query Editor

```
1 SELECT *
2 FROM couchmusic2
3 USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Execute Explain success | elapsed: 1.98ms | execution: 1.96ms | count: 2 | size: 3030 Preferences

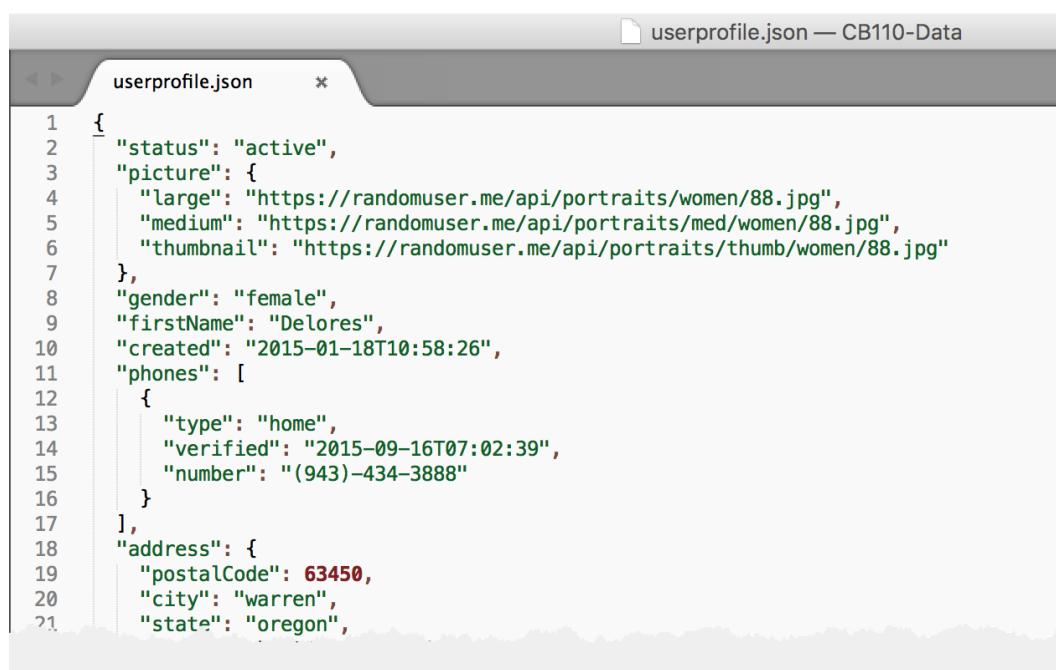
Query Results

JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "couchmusic2": {
4       "address": {
5         "city": "warren",
6         "countryCode": "US",
7         "postalCode": 63450,
8         "state": "oregon",
9         "street": "6174 elgin st"
10      }
11    }
12  ]
13 }
```

B. Select named and nested attributes

8. From the CB110-Data/samples folder, open the *userprofile.json* document. Review the *Userprofile* document attributes and their structure.



9. In *Query Workbench*, modify the prior query to select the *email* and *address* attributes of the specified documents.

```
SELECT email, address
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Notice the structure of the JSON returned by this query: a JSON array of two objects, each with two properties, address and email. Address is itself an object. Email is a string.

Query Editor

```
1 SELECT email, address
2 FROM couchmusic2
3 USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Execute Explain success | elapsed: 6.38ms | execution: 6.36ms | count: 2 | size: 574 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "address": {
4       "city": "warren",
5       "countryCode": "US",
6       "postalCode": 63450,
7       "state": "oregon",
8       "street": "6174 elgin st"
9     },
10    "email": "delores.riley@hotmail.com"
11  },
12  {
13    "address": {
14      "city": "warragul",
15    }
16 }
```

10. Modify the prior query to select the *city* and *countryCode* attributes, while noticing these attributes are nested within the *address* attribute. You should see an error.

```
SELECT city, countryCode
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Query Editor

```
1 SELECT city, countryCode
2 FROM couchmusic2
3 This query contains the following fields not found in the inferred schema for their bucket:
  couchmusic2.city
```

Execute Explain success | elapsed: 1.70ms | execution: 1.68ms | count: 2 | size: 4 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2   {},
3   {}
4 ]
```

11. Modify the prior query to add prefixes, and select instead the `address.city` and `address.countryCode` attributes.

```
SELECT address.city, address.countryCode
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Notice that JSON objects with the selected attributes are returned.

The screenshot shows the Futon interface with the Query Editor at the top. The query is:

- 1 `SELECT address.city, address.countryCode`
- 2 `FROM couchmusic2`
- 3 `USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];`

Below the editor, there are buttons for Execute, Explain, and Preferences. The status bar indicates success with elapsed, execution, count, and size information. The Query Results section shows the JSON output:

```

1 [ 
2   {
3     "city": "warren",
4     "countryCode": "US"
5   },
6   {
7     "city": "warragul",
8     "countryCode": "AU"
9   }
10 ]

```

A red box highlights the JSON array containing two objects. Below the results, there are tabs for JSON, Table, Tree, Plan, and Plan Text.

C. Use aliases, concatenate values (AS, ||), and access document metadata

12. Modify the prior query to select the `firstName` and `lastName` attributes as a single value concatenated using the `||` operator with a single space, along with the `email` attribute.

```
SELECT firstName || " " || lastName, email
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Notice the concatenated value is returned with a system-assigned key of \$1.

The screenshot shows the Futon interface with the Query Editor at the top. The query is:

```
SELECT firstName || " " || lastName, email
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

The results show two documents, each with a system-assigned key (\$1) and an email address:

```

1 [ 
2   {
3     "$1": "Delores Riley",
4     "email": "delores.riley@hotmail.com"
5   },
6   {
7     "$1": "Douglas Moore",
8     "email": "douglas.moore@ymail.com"
9   }
10 ]

```

A red box highlights the key-value pairs for both documents. Below the results, there are tabs for JSON, Table, Tree, Plan, and Plan Text.

13. Modify the prior query, using AS to alias the concatenated value as *fullName*.

```
SELECT firstName || " " || lastName AS fullName, email
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Notice that *email* and *fullName* return in alphabetical order, even though *fullName* appears prior to *email* in the query. N1QL returns all attributes in an object alphabetically.

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 {
3   "email": "delores.riley@hotmail.com",
4   "fullName": "Delores Riley"
5 },
6 {
7   "email": "douglas.moore@ymail.com",
8   "fullName": "Douglas Moore"
9 }
10 ]
```

14. Modify the prior query to use the *META()* function to select the document metadata, aliased as an attribute named *metadata*, in addition to the previously selected attributes.

```
SELECT firstName || " " || lastName AS fullName, email, META() AS metadata
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Query Editor

← history (13/13) →

```
1 SELECT firstName || " " || lastName AS fullName, email, META() AS metadata
2 FROM couchmusic2
3 USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

Execute Explain success | elapsed: 2.63ms | execution: 2.61ms | count: 2 | size: 686 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 {
3   "email": "delores.riley@hotmail.com",
4   "fullName": "Delores Riley",
5   "metadata": {
6     "cas": 1501179114564091904,
7     "expiration": 0,
8     "flags": 33554438,
9     "id": "userprofile::aahingeffeteness42037",
10    "type": "json"
11  }
12 }]
```

End of Lab

Lab 4 - Indexing specified document attributes and selecting by index values

Objectives

A	Filter queries by attribute values (WHERE)
A	Implement an index for a specific document attribute (CREATE INDEX)
B	Create and use an index for a multiply filtered query (AND)
C	Implement an index for an attribute and filter (CREATE INDEX ... WHERE)

Note, this lab assumes you are already familiar with using the *AND* keyword in SQL, when applying multiple filter clauses to a query.

A. Filter queries by attribute values (WHERE)

1. Select the *address* of a document with the *email* value "delores.riley@hotmail.com".

```
SELECT address
FROM couchmusic2
WHERE email = "delores.riley@hotmail.com";
```

Notice the relatively long execution time for this query, due to the full bucket scan it requires. The specific time will vary by system, but will be longer than necessary.

Query Editor

```
1 SELECT address
2 FROM couchmusic2
3 WHERE email = "delores.riley@hotmail.com";|
```

Execute Explain success | elapsed: 4.16s | execution: 4.16s | count: 1 | size: 235 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2   {
3     "address": {
4       "city": "warren",
5       "countryCode": "US",
6       "postalCode": 63450,
7       "state": "oregon",
8       "street": "6174 elgin st"
9     }
10  }
11 ]
```

B. Implement a secondary index for a specific document attribute (CREATE INDEX)

2. Create a secondary index for *email* attributes in the *couchmusic2* bucket.

```
CREATE INDEX idx_email  
ON couchmusic2(email);
```

Notice this statement returns with a *success* status, but no values other than the typical query response metrics.

The screenshot shows the Futon Query Editor interface. In the Query Editor panel, two lines of code are shown: 'CREATE INDEX idx_email' and 'ON couchmusic2(email);'. The second line is highlighted with a red box. Below the editor are buttons for 'Execute' (which is blue), 'Explain', and a status message 'success | elapsed: 3.25s | execution: 3.25s | count: 0 | size: 0'. To the right is a 'history' button with '(15/15)' and a 'Preferences' button. The Query Results panel below shows a JSON response: '1 [{ "results": [] }]'.

3. Re-run the prior query, selecting *address* attributes by *email* address. You can press the back-arrow button in the query history, and press *Execute*.

```
SELECT address  
FROM couchmusic2  
WHERE email = "delores.riley@hotmail.com";
```

Notice the execution time drops to milliseconds.

The screenshot shows the Futon Query Editor interface. The history bar at the top has a yellow star icon and '(14/15)'. In the Query Editor panel, three lines of code are shown: '1 SELECT address', '2 FROM couchmusic2', and '3 WHERE email = "delores.riley@hotmail.com";'. Below the editor are buttons for 'Execute' (which is blue and has a yellow star icon), 'Explain', and a status message 'success | elapsed: 2.09ms | execution: 2.07ms | count: 1 | size: 235'. The Query Results panel shows a JSON response: '1 [{ "address": { "city": "warren", }}]'

C. Create and use an index on a multiply filtered query (AND)

4. Index the *address.postalCode* attributes of *couchmusic2*.

```
CREATE INDEX idx_postalCode  
ON couchmusic2(address.postalCode);
```

5. Select *firstName*, *lastName*, and *address* attributes where the *postalCode* is 63450 and *firstName* is "Roger".

```
SELECT firstName, lastName, address  
FROM couchmusic2  
WHERE address.postalCode = 63450  
AND firstName = "Roger";
```

Notice the query utilizes the available *address.postalCode* index to provide millisecond response, even though the *firstName* attribute is not indexed.

Query Editor

```
1 SELECT firstName, lastName, address  
2 FROM couchmusic2  
3 WHERE address.postalCode = 63450  
4 AND firstName = "Roger";
```

Execute Explain success | elapsed: 4.31ms | execution: 4.30ms | count: 1 | size: 312 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [  
2 {  
3   "address": {  
4     "city": "Clane",  
5     "countryCode": "IE",  
6     "postalCode": 63450,  
7     "state": "california",  
8     "street": "3587 patrick street"  
9   },  
10   "firstName": "Roger",  
11   "lastName": "Smythe"  
12 }  
13 ]
```

D. Implement an index for an attribute and filter (CREATE INDEX ... WHERE)

6. Index the *address.state* attributes of *couchmusic2* where the *status* attribute of a document is "active".

```
CREATE INDEX idx_state_active  
ON couchmusic2(address.state)  
WHERE status = "active";
```

7. Select `email` attributes where `address.state` is "oregon" (note, the `address.state` values are all lowercase. Case insensitive searches are taught in a later Lab.)

```
SELECT email
FROM couchmusic2
WHERE address.state = "oregon";
```

Notice the index is not used, and it takes several seconds to return 225 documents.

Query Editor

```
1 SELECT email
2 FROM couchmusic2
3 WHERE address.state = "oregon";
```

Execute Explain success | elapsed: 4.16s | execution: 4.16s | count: 225 | size: 13079

Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "email": "delores.riley@hotmail.com"
4   },
5   {
6     "email": "lily.nelson@juno.com"
7   },
8   {
9     "email": "lesa.west@ymail.com"
```

8. Modify the prior query to filter both by `address.state` and where `status` is "active".

```
SELECT email
FROM couchmusic2
WHERE address.state = "oregon"
AND status = "active";
```

Notice the index is now used to return 164 results in milliseconds.

Query Editor

```
1 SELECT email
2 FROM couchmusic2
3 WHERE address.state = "oregon"
4 AND status = "active";
```

Execute Explain success | elapsed: 9.87ms | execution: 9.85ms | count: 164 | size: 9509

Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "email": "delores.riley@hotmail.com"
4   },
5   {
```

End of Lab

Lab 5 - Querying ranges, ordering results, and explaining query details

Objectives

A	Query a value range and order the results (AND, ORDER BY)
B	Introspect (EXPLAIN) a query to determine its index use
C	Verify available indexes in the Couchbase console

A. Query a value range and order the results (AND, ORDER BY)

1. From the *CB110-Data/samples* folder, open the *country.json* document. Review the *Country* document attributes and their structure.

```
country.json
1 {
2   "gdp": 40214,
3   "updated": "2015-10-01T07:35:13",
4   "region-number": 39,
5   "name": "Andorra",
6   "countryCode": "AD",
7   "type": "country",
8   "population": 80792
9 }
```

2. Select the *name* and *population* attributes of *country* documents, only, where the *population* is between one and five million, inclusive.

```
SELECT name, population
FROM couchmusic2
WHERE population >= 1000000 AND population <= 5000000
AND type = "country";
```

Notice the documents appear ordered by name. This is not guaranteed behavior.

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2 {
3   "name": "Albania",
4   "population": 3195196
5 },
6 {
7   "name": "Armenia",
8   "population": 2989427
9 },
10 {
11   "name": "Bosnia and Herzegovina",
12   "population": 3816291
13 }
```

3. Modify the query to sequence the results in descending order, by population.

```
SELECT name, population
FROM couchmusic2
WHERE population >= 1000000 AND population <= 5000000
AND type = "country"
ORDER BY population DESC;
```

Query Results

JSON Table Tree Plan Plan Text

```

1- [
2-   {
3-     "name": "Central African Republic",
4-     "population": 4806375
5-   },
6-   {
7-     "name": "Ireland",
8-     "population": 4732269
9-   },
10-  {
11-    "name": "Congo",
12-    "population": 4679663

```

B. Introspect (EXPLAIN) a query to determine its index use

4. Click the *Explain* button, examine the execution plan for this query.

Note the `#operator` value, which in this case is *PrimaryScan*, indicating that the primary index will be used to perform a full bucket scan to complete this query.

Query Editor

← history (22/22) →

```
1 SELECT name, population
2 FROM couchmusic2
3 WHERE population >= 1000000 AND population <= 5000000
4 AND type = "country"
5 ORDER BY population DESC;
```

Execute Explain Explain success | elapsed: | execution: | count: | size:

⚙ Preferences

Query Results

JSON Table Tree Plan Plan Text

```

1- [
2-   {
3-     "plan": {
4-       "#operator": "Sequence",
5-       "~children": [
6-         {
7-           "#operator": "Sequence",
8-           "~children": [
9-             {
10-               "#operator": "PrimaryScan",
11-               "index": "#primary",
12-               "keyspace": "couchmusic2",
13-               "namespace": "default",
14-               "using": "gsi"
15-             },
16-             {
17-               "#operator": "Fetch",

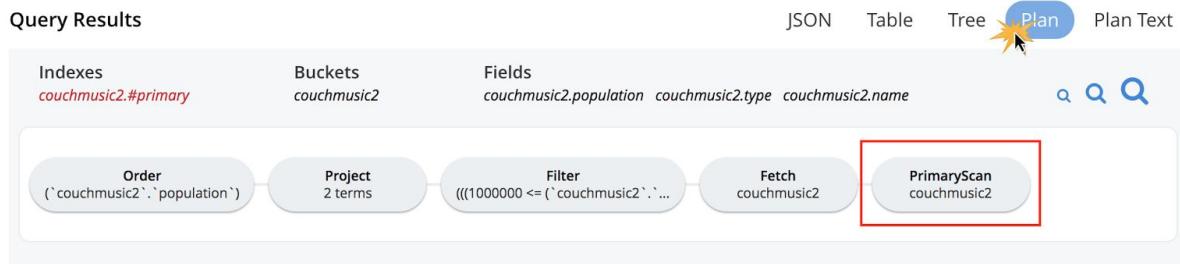
```

5. (Optional) Alternately, you can modify and re-execute the query to EXPLAIN its execution plan.

EXPLAIN

```
SELECT name, population
FROM couchmusic2
WHERE population >= 1000000 AND population <= 5000000
AND type = "country"
ORDER BY population DESC;
```

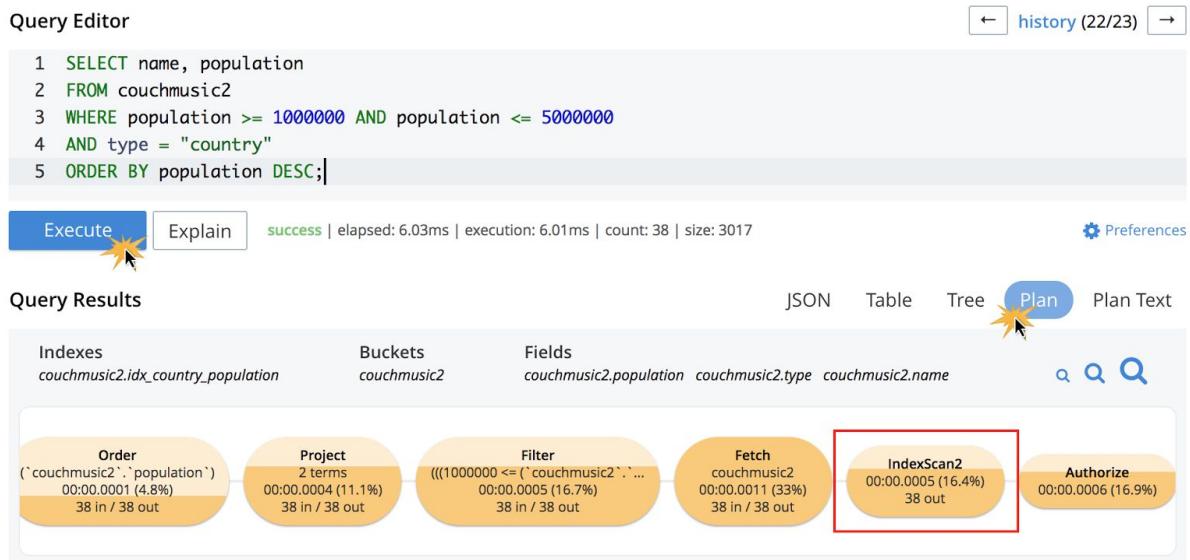
6. Examine the Plan Text and Plan for this query. Notice that the visualized Plan identifies that a PrimaryScan will be used for this query.



7. Index the population attribute of country documents, only.

```
CREATE INDEX idx_country_population
ON couchmusic2(population)
WHERE type = "country";
```

8. Re-execute the prior query, and re-examine its Plan. #operator is now an *IndexScan2*.



6. Review the full structure of the Plan Text. Notice the *index* in use is identified, along with *spans*, indicating the *high* and *low range* being filtered. Notice the bucket name is referred to as *keyspace*.

Query Results

JSON Table Tree Plan Plan Text

```

42
43  " #operator": "IndexScan2",
44  " #stats": {
45    "#itemsOut": 38,
46    "#phaseSwitches": 155,
47    "execTime": "61.125µs",
48    "kernTime": "17.673µs",
49    "servTime": "528.32µs"
50  },
51  "index": "idx_country_population",
52  "index_id": "a708e18f09d9bf",
53  "index_projection": {
54    "primary_key": true
55  },
56  "keyspace": "couchmusic2",
57  "namespace": "default",
58  "spans": [
59    {
60      "exact": true,
61      "range": [
62        {
63          "high": "5000000",
64          "inclusion": 3,
65          "low": "1000000"
66        }
67      ]
68    }
69  }
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165

```

Obviously, there is much more to the Plan text. Comprehensive discussion of N1QL query plans, and strategies for query optimization, are taught in the instructor-led course:

CD210: Couchbase NoSQL Data Modeling, Querying, and Tuning Using N1QL
<http://training.couchbase.com>

C. Verify available indexes in the Couchbase console and by query

1. Index the *status* attribute of *userprofile* documents in *couchmusic2*.

```
CREATE INDEX idx_userprofile_status
ON couchmusic2(status)
WHERE type = "userprofile";
```

2. In the Couchbase console, open the Indexes UI for Global Indexes, and verify the *idx_userprofile_status* index has been created. Click to examine its text.

Buckets
Indexes
 Search
 Query
 XDCR
 Security
 Settings
 Logs

couchmusic2	127.0.0.1:8091	#primary	memory optimized gsi	ready	100%
couchmusic2	127.0.0.1:8091	idx_country_population	memory optimized gsi	ready	100%
couchmusic2	127.0.0.1:8091	idx_email	memory optimized gsi	ready	100%
couchmusic2	127.0.0.1:8091	idx_postalCode	memory optimized gsi	ready	100%
couchmusic2	127.0.0.1:8091	idx_state_active	memory optimized gsi	ready	100%
couchmusic2	127.0.0.1:8091	idx_userprofile_status	memory optimized gsi	ready	100%

Definition
 CREATE INDEX `idx_userprofile_status` ON `couchmusic2`(`status`) WHERE (`type` = "userprofile")

3. In the Query Workbench, query all *name*, *index_key*, and *condition* attributes in the *system:indexes* keyspace to view these attributes of all 6 indexes you have created for *couchmusic2*. Notice which have *condition* attributes, and consider why this is so.

```
SELECT name, index_key, condition
FROM system:indexes
WHERE keyspace_id = "couchmusic2";
```

Query Editor ← history (25/25) →

```
1 SELECT name, index_key, condition
2 FROM system:indexes
3 WHERE keyspace_id = "couchmusic2";
```

Execute Explain success | elapsed: 42.53ms | execution: 42.51ms | count: 6 | size: 857 Preferences

Query Results JSON Table Tree Plan Plan Text

```
1 [
2 {
3   "condition": "(`type` = \"userprofile\")",
4   "index_key": [
5     "`status`"
6   ],
7   "name": "idx_userprofile_status"
8 },
9 {
10   "condition": "(`type` = \"country\")",
11   "index_key": [
12     "`population`"
13   ],
14   "name": "idx_country_population"
```

End of Lab

Lab 6 - Counting aggregate values, and selecting distinct results using wildcards

Objectives

A	Count selected values (COUNT)
B	Observe a covering index
C	Using wildcard filters (LIKE) to select distinct values (DISTINCT)

A. Count selected values (COUNT)

3. Use the COUNT keyword to count all *userprofiles* with "active" status in *couchmusic2*, AS *active_profiles*. Ensure your filter is written to match the index created above.

```
SELECT COUNT(*) AS active_profiles
FROM couchmusic2
WHERE type = "userprofile"
AND status = "active";
```

Query Editor

```
1 SELECT COUNT(*) AS active_profiles
2 FROM couchmusic2
3 WHERE type = "userprofile"
4 AND status = "active";
```

← history (26/26) →

Execute Explain success | elapsed: 13.44ms | execution: 13.42ms | count: 1 | size: 48 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2 {
3   "active_profiles": 37457
4 }
5 ]
```

B. Observe a covering index

- Examine the Plan Text for the prior query.

Notice the `idx_userprofile_status` index has been used as a Covering Index - an index which directly supplies the information needed to satisfy a query - by providing the entries necessary to provide a COUNT(*) of the underlying documents.

Query Results

JSON Table Tree Plan **Plan Text**

```
31
32     "children": [
33         {
34             "#operator": "IndexCountScan2",
35             "#stats": {
36                 "#itemsOut": 1,
37                 "#phaseSwitches": 3,
38                 "execTime": "9.339822ms",
39                 "kernTime": "5.007µs"
40             },
41             "covers": [
42                 "cover ((`couchmusic2`.`status`))",
43                 "cover ((meta(`couchmusic2`).`id`))"
44             ],
45             "filter_covers": [
46                 "cover ((`couchmusic2`.`type`))": "userprofile"
47             },
48             "index": "idx_userprofile_status",
49             "index_id": "1f6aff94bf765249",
50             "keyspace": "couchmusic2",
51             "namespace": "default",
52             "scope": "public"
53         }
54     ]
55 }
```

C. Using wildcard filters (LIKE) to select distinct values (DISTINCT)

- In Couchbase console, verify that you created the `idx_email` index in a prior Lab.

Indexes	couchmusic2	127.0.0.1:8091	#primary	memory optimized gsi	ready	100%
Search	couchmusic2	127.0.0.1:8091	idx_country_population	memory optimized gsi	ready	100%
Query	couchmusic2	127.0.0.1:8091	idx_email	memory optimized gsi	ready	100%

Note, if this index does not already exist, create it using the following statement.

```
CREATE INDEX idx_email
ON couchmusic2(email);
```

5. To determine in which countries users are using Hotmail, select distinct `countryCode` values in `address` objects within `userprofile` documents, where the `email` attribute contains the value "`@hotmail.com`".

```
SELECT DISTINCT address.countryCode
FROM couchmusic2
WHERE email LIKE "%hotmail.com";
```

Query Editor

← history (27/27) →

```
1 SELECT DISTINCT address.countryCode
2 FROM couchmusic2
3 WHERE email LIKE "%hotmail.com";
```

Execute

Explain

success | elapsed: 995.74ms | execution: 995.72ms | count: 8 | size: 344

⚙ Preferences

Query Results

JSON

Table

Tree

Plan

Plan Text

```
1 [ 
2   {
3     "countryCode": "FI"
4   },
5   {
6     "countryCode": "NL"
7   },
8   {
9     "countryCode": "GB"
10 }
```

End of Lab

Lab 7 - Selecting for missing attributes, grouping results by attributes, and considering index optimization

Objectives

A	Select documents missing a specified attribute (IS MISSING, IS NOT MISSING)
B	Group attributes and order results to generate summary values (GROUP BY)
C	Consider indexing and optimization

A. Select documents missing a specified attribute (IS MISSING)

1. Select the *firstName*, *lastName*, and *address* object of all *userprofile* documents in *couchmusic2*, where no *gender* attribute has been provided.

```
SELECT firstName, lastName, email
FROM couchmusic2
WHERE type = "userprofile"
AND gender IS MISSING;
```

Query Editor ← history (28/28) →

```
1 SELECT firstName, lastName, email
2 FROM couchmusic2
3 WHERE type = "userprofile"
4 AND gender IS MISSING;|
```

Execute Explain success | elapsed: 5.58s | execution: 5.58s | count: 24904 | size: 3181071 Preferences

Query Results JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "email": "gerry.oliver@hotmail.com",
4     "firstName": "Gerry",
5     "lastName": "Oliver"
6   },
7   {
8     "email": "malone.colin@yahoo.com",
9     "firstName": "Malone",
10    "lastName": "Colin"
11  },
12  {
```

- Count the number of *userprofile* documents where gender is not missing.

```
SELECT COUNT(gender) as have_gender
FROM couchmusic2
WHERE type = "userprofile"
AND gender IS NOT MISSING;
```

Would an index (aka, a summary table of gender values mapped to corresponding document keys) help determine which *userprofile* documents are missing this value? Or, is a full bucket scan functionally required to determine missing attributes?

B. Group attributes and order results to generate summary values (GROUP BY)

- How many user profiles exist for each country? Select the *countryCode*, and count the *userprofile* documents in general (*), in *couchmusic1*. Group the results by *countryCode*, and order the results in descending order by count.

```
SELECT address.countryCode, COUNT(*) AS user_count
FROM couchmusic2
GROUP BY address.countryCode
ORDER BY user_count DESC;
```

Notice the results for this query - unrestricted by document type - implicitly identify how many documents in *couchmusic2* do not have an *address.countryCode* attribute.

The screenshot shows a database query editor interface. At the top, there is a 'Query Editor' section containing the SQL query:

```
1 SELECT address.countryCode, COUNT(*) AS user_count
2 FROM couchmusic2
3 GROUP BY address.countryCode
4 ORDER BY user_count DESC;
```

Below the query editor, there are buttons for 'Execute' and 'Explain'. The status bar indicates 'success | elapsed: 4.24s | execution: 4.24s | count: 9 | size: 644'. To the right, there are links for 'history (32/32)' and 'Preferences'.

At the bottom, there is a 'Query Results' section. It includes tabs for 'JSON', 'Table', 'Tree', 'Plan', and 'Plan Text'. The 'JSON' tab is selected, showing the results of the query as a JSON array:

```
1 [
2   {
3     "user_count": 163082
4   },
5   {
6     "countryCode": "FI",
7     "user_count": 9559
8   },
9   {
10    "countryCode": "NL",
11    "user_count": 7010
12 }
```

The value '163082' is highlighted with a red box.

C. Consider indexing and optimization

4. Modify the prior query to count and group only *userprofile* documents. Run the query, and consider the difference in output, and execution time.

```
SELECT address.countryCode, COUNT(*) AS user_count
FROM couchmusic2
WHERE type = "userprofile"
GROUP BY address.countryCode
ORDER BY user_count DESC;
```

Would indexing *type* improve execution time for this query? By how much?

5. (Optional) Index the *type* attribute in *couchmusic2*, re-run the prior query, and compare its execution time with the prior. Consider how many different *type* values are present in *couchmusic2* (userprofile, track, playlist, country, sub-region), and how the cardinality of values for an indexed attribute relates to index effectiveness in reducing execution time.

End of Lab

Lab 8 - Indexing and selecting based on values in JSON string and object arrays

Objectives

A	Selecting where a specific value exists within an array (WHERE, NOT, IN)
B	Indexing a JSON array, and using the newly created index
C	Selecting where any/every value in an object array satisfies an expression (WHERE, SATISFIES, ANY, EVERY, END)

A. Selecting where a specified value exists within an array (WHERE, NOT, IN)

1. Select and review the structure of a *userprofile* document - which itself is a JSON object, returned by N1QL in an array of objects, and the various nested objects and array each document may contain. What if you needed to track multiple addresses per User?

```
SELECT *
FROM couchmusic2
WHERE type = "userprofile"
LIMIT 1;
```

```
1 {
2   "status": "active",
3   "picture": {
4     "large": "https://randomuser.me/api/portraits/women/88.jpg",
5     "medium": "https://randomuser.me/api/portraits/med/women/88.jpg",
6     "thumbnail": "https://randomuser.me/api/portraits/thumb/women/88.jpg"
7   },
8   "gender": "female",
9   "firstName": "Delores",
10  "created": "2015-01-18T10:58:26",
11  "phones": [
12    {
13      "type": "home",
14      "verified": "2015-09-16T07:02:39",
15      "number": "(943)-434-3888"
16    }
17  ],
18  "updated": "2015-08-25T10:28:37",
19  "username": "aahingefteteness42037",
20  "address": {
21    "postalCode": 63450,
22    "city": "warren",
23    "state": "oregon",
24    "street": "6174 elgin st",
25    "countryCode": "US"
26  },
27  "dateOfBirth": "1983-07-12",
28  "pwd": "636172626fge",
29  "favoriteGenres": [
30    "Classical Crossover",
31    "Contemporary Blues",
32    "French Pop",
33    "Progressive Bluegrass"
34  ],
35  "lastName": "Riley",
36  "title": "Mrs",
37  "type": "userprofile",
38  "email": "delores.riley@hotmail.com"
39 }
```

For an introduction to data modeling in JSON, including structural options in document design, take this free Online Training course at <http://training.couchbase.com/online>.

CB105 - Introduction to Data Modeling in JSON

2. Index *userprofile* documents by gender.

```
CREATE INDEX idx_userprofile_gender
ON couchmusic2(gender)
WHERE type = "userprofile";
```

3. Count the *userprofile* document which list gender as "*female*" and "*folk*" as a favorite genre in the *favoriteGenres* array.

```
SELECT COUNT(*) AS female_folk_fans
FROM couchmusic2
WHERE type = "userprofile"
AND 'Folk' IN favoriteGenres
AND gender = "female";
```

Query Editor

```
1 SELECT COUNT(*) AS female_folk_fans
2 FROM couchmusic2
3 WHERE type = "userprofile"
4 AND 'Folk' IN favoriteGenres
5 AND gender = "female";
```

Execute Explain success | elapsed: 428.73ms | execution: 428.71ms | count: 1 | size: 47 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 { 
3   "female_folk_fans": 166
4 } 
5 ]
```

4. Modify the prior query to count female users who do not like "*folk*".

```
SELECT COUNT(*) AS female_NOT_folk_fans
FROM couchmusic2
WHERE type = "userprofile"
AND 'Folk' NOT IN favoriteGenres
AND gender = "female";
```

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 { 
3   "female_NOT_folk_fans": 12376
4 } 
5 ]
```

B. Indexing a JSON array, and using the newly created index (DISTINCT ARRAY, FOR .. IN .. END)

4. Index distinct values in the *favoriteGenres* array where document type is "userprofile".
Use "genre" as the name of the index variable.

```
CREATE INDEX idx_favoriteGenres
ON couchmusic2 (
    DISTINCT ARRAY genre
    FOR genre IN favoriteGenres END
)
WHERE type = "userprofile";
```

5. Select address.state and email for any userprofile which lists "Folk" as a favorite genre.

```
SELECT address.state, email
FROM couchmusic2
WHERE type = "userprofile"
AND 'Folk' IN favoriteGenres;
```

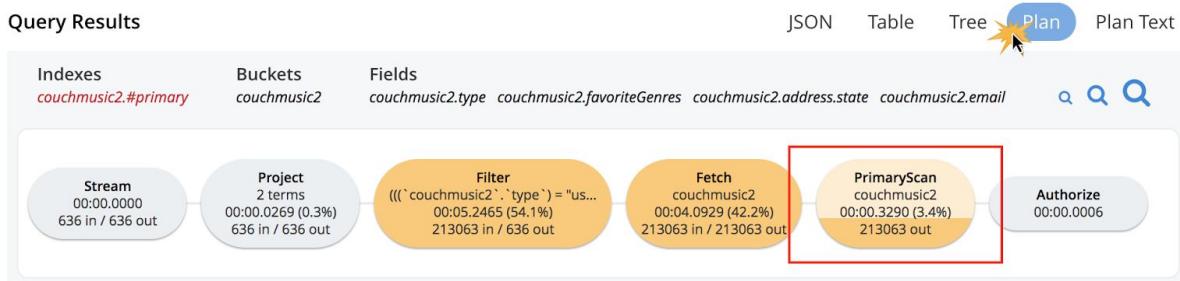
Notice the slow, multiple second execution time, indicating no index was used.

The screenshot shows a query interface with the following details:

- Query text: `4 AND 'folk' IN favoriteGenres;`
- Status: success | elapsed: 5.41s | execution: 5.41s | count: 636 | size: 59915
- Buttons: Execute, Explain, Preferences
- Results tab: Query Results, JSON, Table, Tree, Plan, Plan Text (Plan is selected)
- Result content (JSON format):

```
1- [
2-   {
3-     "email": "konsta.saarinen@cox.net",
4-     "state": "south karelia"
5-   },
6-   {
7-     "email": "luukas.elo@cox.net",
8-     "state": "south karelia"
9-   }
]
```

6. Display the query Plan, to verify no index was used.



7. Modify the prior query to use expression syntax to select document where ANY genre IN favoriteGenres SATISFIES the expression `genre = "Folk"`.

```
SELECT address.state, email
FROM couchmusic2
WHERE type = "userprofile"
```

```

3 WHERE type = "userprofile"
4 AND ANY genre IN favoriteGenres
5 SATISFIES genre = "Folk" END;

```

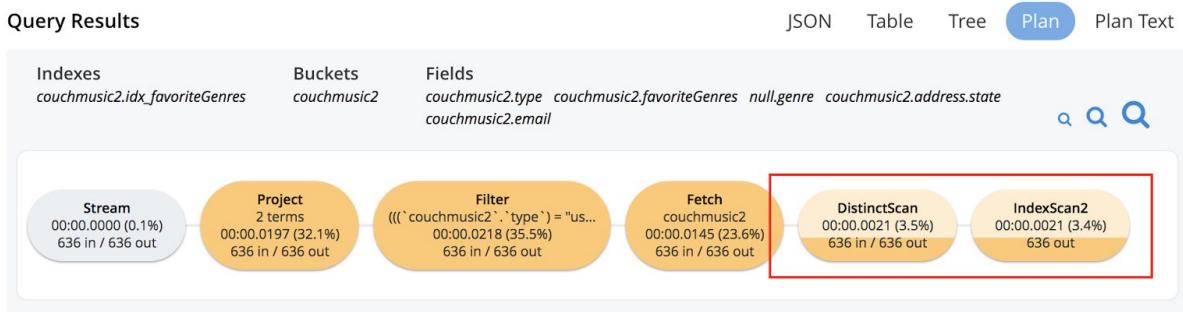
Notice the *idx_favoriteGenres* index is now used.

```

1 [
2 {
3   "email": "konsta.saarinen@cox.net",
4   "state": "south karelia"
5 },
6 {
7   "email": "Juukka elo@cox.net"
}

```

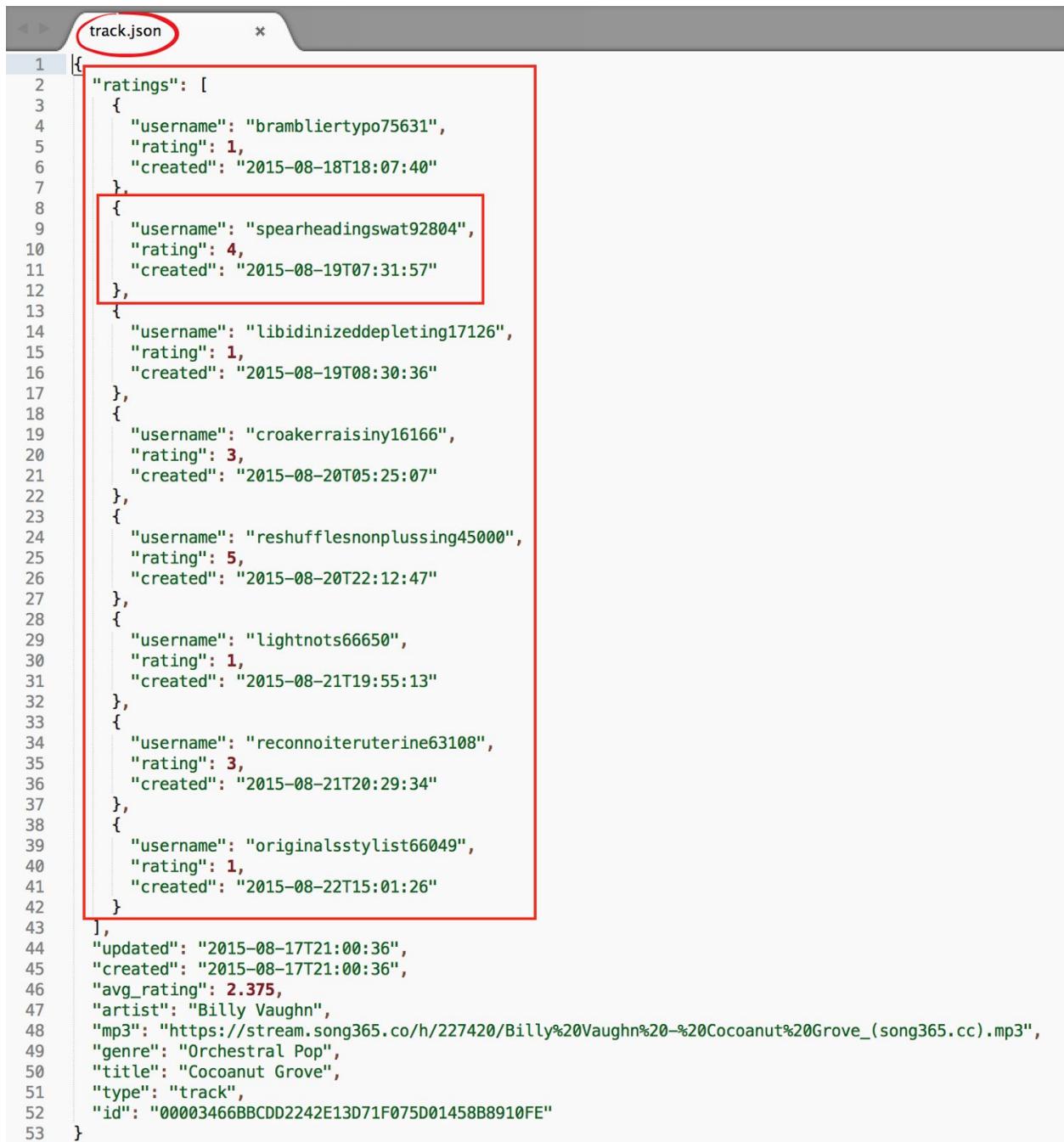
8. Display the query Plan, to verify which indexing was used.



9. (Optional) *Explain* the query, or view its *Plan Text*, to determine the specific indexing and related spans used for this query.

C. Selecting where any/every value in an object array satisfies an expression (WHERE, SATISFIES, ANY, EVERY, END)

9. Open and review the *track.json* sample document from the *CD110-Data/samples* folder. The *ratings* attribute is an array of individual objects, each with its own *rating* attribute, which has an integer value ranging from 1 to 5.



```
track.json
[
  {
    "ratings": [
      {
        "username": "brambliertypo75631",
        "rating": 1,
        "created": "2015-08-18T18:07:40"
      },
      {
        "username": "spearheadingswat92804",
        "rating": 4,
        "created": "2015-08-19T07:31:57"
      },
      {
        "username": "libidinizeddepleting17126",
        "rating": 1,
        "created": "2015-08-19T08:30:36"
      },
      {
        "username": "croakerraissiny16166",
        "rating": 3,
        "created": "2015-08-20T05:25:07"
      },
      {
        "username": "reshufflesnonplussing45000",
        "rating": 5,
        "created": "2015-08-20T22:12:47"
      },
      {
        "username": "lightnots66650",
        "rating": 1,
        "created": "2015-08-21T19:55:13"
      },
      {
        "username": "reconnoiteruterine63108",
        "rating": 3,
        "created": "2015-08-21T20:29:34"
      },
      {
        "username": "originalsstylist66049",
        "rating": 1,
        "created": "2015-08-22T15:01:26"
      }
    ],
    "updated": "2015-08-17T21:00:36",
    "created": "2015-08-17T21:00:36",
    "avg_rating": 2.375,
    "artist": "Billy Vaughn",
    "mp3": "https://stream.song365.co/h/227420/Billy%20Vaughn%20-%20Cocoanut%20Grove_(song365.cc).mp3",
    "genre": "Orchestral Pop",
    "title": "Cocoanut Grove",
    "type": "track",
    "id": "00003466BBCDD2242E13D71F075D01458B8910FE"
  }
]
```

10. Select the *artist*, *title*, *genre*, and *id* from *track* documents, where the results satisfy an expression requiring that a user with the username "conwormish43746" has given the *track* a 5 *rating*.

```
SELECT artist, title, genre, id
FROM couchmusic2
WHERE type = "track"
AND ANY r IN ratings SATISFIES
    r.username = "conwormish43746" AND r.rating = 5
END;
```

Query Results

JSON Table Tree Plan Plan Text

```
1~ [
2~ {
3~   "artist": "Artillery",
4~   "genre": "Heavy Metal",
5~   "id": "649627A6378C46BCA81BE32A5026DE5B17350EEA",
6~   "title": "All For You"
7~ },
8~ {
9~   "artist": "Bobby Darin",
10~  "genre": "Folk Rock",
11~  "id": "B310AF664FF93F8B6BBC152C42DEAED82B0DFFC0",
12~  "title": "I'll Remember April"
13~ }
14~ ]
```

End of Lab

Lab 9 - Using functions and expressions in queries

Objectives

A	Use a function to check array length as a query condition (ARRAY_LENGTH)
B	Use functions to calculate values from document data (MIN, STR_TO_MILLIS)
C	Declare inline array to support using functions in aggregate calculations ([...])
D	Use mathematical operators to calculate values inline, then truncate the result to an integer (- , / , TRUNC)

A. Use a function to check array length as a query condition (ARRAY_LENGTH)

1. Modify the final query of the prior lab to return *track* documents where there is more than one rating, and all ratings have the value of 5.

```
SELECT artist, title, genre, id, ratings
FROM couchmusic2
WHERE type = "track"
AND ARRAY_LENGTH(ratings) >= 3
AND EVERY r IN ratings SATISFIES r.rating = 5 END;
```

Query Results

JSON Table Tree Plan Plan Text

```
1- [
2- {
3-   "artist": "Yummy Bingham",
4-   "genre": "Urban",
5-   "id": "01210A1C923BC8501FEA1416C0E8D3F81539561F",
6-   "ratings": [
7-     {
8-       "created": "2015-08-20T06:38:30",
9-       "rating": 5,
10-      "username": "meticulousobsessed56962"
11-    },
12-    {
13-      "created": "2015-08-21T06:32:30",
14-      "rating": 5,
15-      "username": "marrowschert53500"
16-    },
17-    {
18-      "created": "2015-08-21T20:25:59",
```

B. Use functions to calculate values from document data (MIN, STR_TO_MILLIS)

2. Use the MIN() function to select the earliest *userprofile dateOfBirth* as the *oldest* value.

```
SELECT MIN(dateOfBirth) AS oldest
FROM couchmusic2
WHERE type = "userprofile";
```

Query Editor

```
1 SELECT MIN(dateOfBirth) AS oldest
2 FROM couchmusic2
3 WHERE type = "userprofile";|
```

Execute Explain success | elapsed: 4.19s | execution: 4.19s | count: 1 | size: 46 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 { 
3   "oldest": "1970-01-01"
4 } 
5 ]
```

3. Modify the prior query to use the STR_TO_MILLIS() function to convert the *userprofile dateOfBirth* values, while still selecting the smallest value, to determine the *oldest* user (whose birth date is closest to the epoch, and so the smallest number of milliseconds).

```
SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest
FROM couchmusic2
WHERE type = "userprofile";
```

Query Results

JSON Table Tree Plan Plan Text

```
1 [ 
2 { 
3   "oldest": 28800000
4 } 
5 ]
```

C. Declare inline array to support using functions in aggregate calculations ([...])

4. Modify the prior query to use the META() function to return document metadata.

```
SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest, META() AS meta
FROM couchmusic2
WHERE type = "userprofile";
```

Note, the query will fail. When aggregating, all selected attributes must be aggregated.

The screenshot shows the Couchbase Query Editor interface. The top section is the 'Query Editor' where the following SQL-like query is entered:

```
1 SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest, META() AS meta
2 FROM couchmusic2
3 WHERE type = "userprofile";
```

Below the editor, there are several buttons: 'Execute' (blue), 'Explain' (white), and a red-bordered button labeled '500'. To the right of these buttons, it says 'elapsed: 557.838µs | execution: 545.391µs | count: 0 | size: 0'. Further to the right are 'history (52/52)' and a 'Preferences' icon.

The bottom section is the 'Query Results' pane. It displays an error response in JSON format:

```
1 [
2   {
3     "code": 4210,
4     "msg": "Expression must be a group key or aggregate: meta(`couchmusic2`)",
5     "query_from_user": "SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest, META() AS meta\nFROM\n  couchmusic2\nWHERE type = \"userprofile\";"}
```

A red arrow points from the '500' button in the Query Editor to the 'msg' field in the JSON results, highlighting the error message.

5. Modify the prior query to use the MIN() function on an inline array comprised of the *dateOfBirth* attribute and the result of calling META() within the object.

```
SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest,
       MIN( [ dateOfBirth, META() ] ) AS meta
  FROM couchmusic2
 WHERE type = "userprofile";
```

Notice the MIN() function is applied to the first array element: *dateOfBirth*.

The screenshot shows a database query editor interface. At the top, there is a 'Query Editor' section containing the following SQL code:

```
1 SELECT MIN(STR_TO_MILLIS(dateOfBirth)) AS oldest,
2       MIN( [ dateOfBirth, META() ] ) AS meta
3   FROM couchmusic2
4 WHERE type = "userprofile";
```

The second line, which contains the MIN() function, is highlighted with a red box and has a red arrow pointing down to the corresponding JSON output in the 'Query Results' section below.

Below the query editor, there are buttons for 'Execute' (highlighted in blue), 'Explain', and 'Preferences'. The status bar indicates 'success | elapsed: 4.78s | execution: 4.78s | count: 1 | size: 366'.

The 'Query Results' section displays the JSON output of the query. The JSON structure is as follows:

```

1 [
2   {
3     "meta": [
4       {
5         "dateOfBirth": "1970-01-01",
6         "meta": {
7           "cas": 1501179114852384768,
8           "expiration": 0,
9           "flags": 33554438,
10          "id": "userprofile::coagulatelassie9653",
11          "type": "json"
12        }
13      ],
14      "oldest": 28800000
15    }
]

```

D. Use math operators to calculate inline, then truncate to an integer (- , / , TRUNC)

6. Modify the prior query to subtract the birth value, in ms, from the current date, in ms.

```
SELECT (CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) AS age_in_ms,
       MIN( [ dateOfBirth, META() ] ) AS meta
  FROM couchmusic2
 WHERE type = "userprofile";
```

Query Editor

```
1 SELECT (CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) AS age_in_ms,
2      MIN( [ dateOfBirth, META() ] ) AS meta
3  FROM couchmusic2
4 WHERE type = "userprofile";
```

Execute Explain success | elapsed: 4.69s | execution: 4.69s | count: 1 | size: 378 Preferences

Query Results

JSON Table Tree Plan Plan Text

1	[
2	{
3	"age_in_ms": 1501247154036.413,
4	"meta": [
5	"1970-01-01",
6	{
7	"cas": 1501179114852384768,
8	"expiration": 0,
9	"flags": 33554438,
10	"id": "userprofile::coagulatelassie9653",
11	"type": "json"
12	}
13]
14	}
15]

7. Modify to calculate age in years, by dividing the result by 3.1536e10 (one year in ms).

```
SELECT (CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) / 3.1536e10 AS age_in_years,
       MIN( [ dateOfBirth, META() ] ) AS meta
  FROM couchmusic2
 WHERE type = "userprofile";
```

Query Editor

```
1 SELECT (CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) / 3.1536e10 AS age_in_years,
2      MIN( [ dateOfBirth, META() ] ) AS meta
3  FROM couchmusic2
4 WHERE type = "userprofile";|
```

Execute Explain success | elapsed: 4.74s | execution: 4.74s | count: 1 | size: 381 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1- [ {
2-   {
3-     "age_in_years": 47.60423617487606,
4-     "meta": [
5-       "1970-01-01",
6-       {
7-         "cas": 1501179114852384768,
8-         "expiration": 0,
9-         "flags": 33554438,
10-        "id": "userprofile::coagulatelassie9653",
11-        "type": "json"
12-       }
13-     ]
14-   }
15- }
```

8. Last, modify to truncate the final value to an integer.

```
SELECT TRUNC((CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) / 3.1536e10)
       AS age_of_oldest_userprofile,
       MIN( [ dateOfBirth, META() ] ) AS meta
FROM couchmusic2
WHERE type = "userprofile";
```

Query Editor ← history (56/56) →

```
1 SELECT TRUNC((CLOCK_MILLIS() - MIN(STR_TO_MILLIS(dateOfBirth))) / 3.1536e10)
2     AS age_of_oldest_userprofile,
3     MIN( [ dateOfBirth, META() ] ) AS meta
4 FROM couchmusic2
5 WHERE type = "userprofile";
```

Execute Explain success | elapsed: 4.88s | execution: 4.88s | count: 1 | size: 379 ⚙ Preferences

Query Results JSON Table Tree Plan Plan Text

```
1 [
2   {
3     "age_of_oldest_userprofile": 47,
4     "meta": [
5       {
6         "dateOfBirth": "1970-01-01",
7         "meta": {
8           "cas": 1501179114852384768,
9           "expiration": 0,
10          "flags": 33554438,
11          "id": "userprofile::coagulatelassie9653",
12          "type": "json"
13        }
14      }
15    ]
}
```

End of Lab

Lab 10 - Manipulating data using N1QL DML

Objectives

A	INSERT a new document
B	UPSERT and UPDATE new and existing documents
C	DELETE documents by value or key

A. *INSERT a new userprofile document*

1. Insert a new *userprofile* document to *couchmusic2*, using *userprofile::aaa-oscar-orange* as its key, and the values shown below. Note, as expectable, INSERT returns metadata, including execution time, but no results.

```
INSERT INTO couchmusic2 (KEY, VALUE)
VALUES ("userprofile::aaa-oscar-orange",
{
  "address": {
    "city": "Portland",
    "countryCode": "US",
    "postalCode": 97203,
    "state": "Oregon"
  },
  "dateOfBirth": "1969-01-23",
  "email": "oscar.orange@gmx.com",
  "favoriteGenres": [
    "Post Punk",
    "Contemporary Blues",
    "Ambient"
  ],
  "firstName": "Oscar",
  "gender": "male",
  "lastName": "Orange",
  "phones": [
    {
      "number": "(503)-555-1222",
      "type": "home",
      "verified": "2016-05-07T11:11:23"
    }
  ],
  "status": "active",
  "title": "Mr",
  "type": "userprofile",
  "updated": "2016-05-23T07:23:32",
  "username": "aaa-oscar-orange"
});
```

2. Execute this query a second time, and notice the duplicate key error.

The screenshot shows the Futon interface with the "Execute" tab selected. The results pane displays an array with one element, which is an object containing a code of 12009 and a msg indicating a DML error due to a duplicate key. A red box highlights this error message.

```

1 [ ]
2 {
3   "code": 12009,
4   "msg": "DML Error, possible causes include CAS mismatch or concurrent modificationFailed to perform
      insert - cause: Duplicate Key userprofile::aaa-oscar-orange"
5 }
6 ]

```

B. UPSERT and UPDATE new and existing documents

3. Modify the INSERT statement to an UPSERT statement.
4. Execute the query a third time, and notice it now succeeds.

```

UPSERT INTO couchmusic2 (KEY, VALUE)
VALUES ("userprofile::aaa-oscar-orange",
{
  "address": {
    "city": "Portland",
    "countryCode": "US",
    "postalCode": 97203,
    "state": "Oregon"
  },
  ...
}
);

```

The screenshot shows the Futon Query Editor with the query `UPSERT INTO couchmusic2 (KEY, VALUE) VALUES ("userprofile::aaa-oscar-orange", { ... })` highlighted. Below the editor, the status bar shows "success" with a red border, indicating the query was executed successfully.

The screenshot shows the Futon Query Results pane. The results are displayed in JSON format as an array with one element, which is an object containing a key and a value object. The status bar at the bottom shows "success" with a red border.

```

1 [
2   {
3     "key": "userprofile::aaa-oscar-orange",
4     "value": {
5       "address": {
6         "city": "Portland",
7         "countryCode": "US",
8         "postalCode": 97203,
9         "state": "Oregon"
10        }
11      }
12    }
13  ]

```

5. Modify the document attributes as shown in bold below.

```
UPSERT INTO couchmusic2 (KEY, VALUE)
VALUES ("userprofile::aaa-betty-blue", 
{
  "address": {
    "city": "Portland",
    "countryCode": "US",
    "postalCode": 97203,
    "state": "Oregon"
  },
  "dateOfBirth": "1969-01-23",
  "email": "betty.blue@gmx.com",
  "favoriteGenres": [
    "Post Punk",
    "Contemporary Blues",
    "Ambient"
  ],
  "firstName": "Betty",
  "gender": "female",
  "lastName": "Blue",
  "phones": [
    {
      "number": "(503)-555-1222",
      "type": "home",
      "verified": "2016-05-07T11:11:23"
    }
  ],
  "status": "active",
  "title": "Ms",
  "type": "userprofile",
  "updated": "2016-05-23T07:23:32",
  "username": "aaa-betty-blue"
}
);
```

6. Execute the query.

7. In the Data Buckets UI, filter the documents by *userprofile* and notice you now have two documents. Because the key did not exist, a new document is created.

The screenshot shows the Couchbase Data Buckets interface. On the left, a sidebar lists navigation options: Dashboard, Servers, Buckets, Indexes, Search, Query, XDCR, and Security. The 'Buckets' option is selected. In the main area, a dropdown menu is open, showing the bucket name 'couchmusic2'. To the right of the dropdown is a search bar with the placeholder 'filter: ?startkey=%22userprofile%22&skip=0&include_docs=true&limit=11'. Below the search bar, there are input fields for 'startkey' (containing 'userprofile') and 'endkey' (empty), and a checked checkbox for 'inclusive_end'. At the bottom right are 'Reset' and 'Save' buttons. A red box highlights the 'userprofile' value in the 'startkey' field. Another red box highlights the list of IDs in the results table, which includes 'userprofile::aaa-betty-blue' and 'userprofile::aaa-oscar-orange'. The status bar at the bottom displays the document content: {"status":"active", "picture":{"large":"https://randomuser.me/api/po...".

8. Index the *username* attribute of *userprofile* documents in *couchmusic2*.

```
CREATE INDEX idx_userprofile_username
ON couchmusic2(username)
WHERE type = "userprofile";
```

9. Select the *firstName*, *lastName*, and *email* attributes of *userprofile* documents where the *username* is *aaa-oscar-orange*.

```
SELECT firstName, lastName, email
FROM couchmusic2
WHERE type = "userprofile"
AND username = "aaa-oscar-orange";
```

10. Update the *email* for this *username*, and update the *updated* timestamp, as shown:

```
UPDATE couchmusic2
SET   email = "oscar.orange2@gmx.com",
      updated = NOW_STR()
WHERE type = "userprofile"
AND username = "aaa-oscar-orange";
```

11. Select all attributes for the *userprofile* with this *username*, to *confirm* the update.

```
SELECT *
FROM couchmusic2
WHERE type = "userprofile"
AND username = "aaa-oscar-orange";
```

C. *DELETE* documents by value or key

12. Delete *userprofile* documents with this *username*.

```
DELETE
FROM couchmusic2
WHERE type = "userprofile"
AND username = "aaa-oscar-orange";
```

13. Delete *userprofile* documents which have the key *userprofile::aaa-betty-blue*.

```
DELETE
FROM couchmusic2
USE KEYS ("userprofile::aaa-betty-blue");
```

14. In the Data Buckets UI, verify the Userprofiles added in this lab have been deleted.

The screenshot shows the Couchbase Data Buckets UI. On the left, there is a dropdown menu set to "couchmusic2". To its right is a search bar with the placeholder "filter: ?startkey=%22userprofile%22&skip=0&include_docs=true&limit=11". Below the search bar, there are two input fields: "startkey" containing "userprofile" and "endkey" which is empty. There is also a checked checkbox labeled "inclusive_end". At the bottom right are "Reset" and "Save" buttons.

End of Lab

Lab 11 - Joining documents by embedded key references

Objectives

A	Join documents by prefixing an embedded document key
B	Join documents by prefixing values from an embedded key array
C	Restructure selected data as an inline object

A. Join documents based on an embedded document key

15. Select the `title`, `firstName`, `lastName`, and `email` from `couchmusic2`.
16. Alias `couchmusic2` as `u` (for references to `userprofile` documents in this query.)
17. Prefix the selected attributes with the `u` alias.
18. Add a USE KEYS clause to select the key `"userprofile::abandoninghouseclean34190"`, and then run the query.

```
SELECT u.firstName, u.lastName, u.email
FROM couchmusic2 AS u
USE KEYS "userprofile::abandoninghouseclean34190";
```

Query Editor ← history (66/66) →

```
1 SELECT u.firstName, u.lastName, u.email
2 FROM couchmusic2 AS u
3 USE KEYS "userprofile::abandoninghouseclean34190";|
```

Execute Explain success | elapsed: 2.11ms | execution: 2.09ms | count: 1 | size: 117 ⚙ Preferences

Query Results JSON Table Tree Plan Plan Text

```
1 [|
2   {
3     "email": "leo.beck@email.com",
4     "firstName": "Leo",
5     "lastName": "Beck"
6   }
7 ]
```

19. In the Couchbase console, open the *couchmusic2* bucket, and review the key pattern used for Country documents. Notice the literal prefix "*country::*" precedes a *countryCode* value.

```

country::AD
Delete Save As... Save
1 {
2   "gdp": 40214,
3   "updated": "2015-10-01T07:35:13",
4   "region-number": 39,
5   "name": "Andorra",
6   "countryCode": "AD",
7   "type": "country",
8   "population": 80792
9 }

```

20. Recall that Userprofile documents include an *address.countryCode* attribute.

```

userprofile::aahingeffeteness42037
Delete Save As... Save
1 {
2   "status": "active",
3   "picture": {
4     "large": "https://randomuser.me/api/portr",
5     "medium": "https://randomuser.me/api/portr",
6     "thumbnail": "https://randomuser.me/api/pic",
7   },
8   "gender": "female",
9   "firstName": "Delores",
10  "created": "2015-01-18T10:58:26",
11  "phones": [
12    {
13      "type": "home",
14      "verified": "2015-09-16T07:02:39",
15      "number": "(943)-434-3888"
16    }
17  ],
18  "address": {
19    "postalCode": 63450,
20    "city": "warren",
21    "state": "oregon",
22    "street": "6174 elgin st",
23    "countryCode": "US"
24  },
25  "updated": "2015-08-25T10:28:37",

```

21. Join the *couchmusic2* bucket AS *c* (for references to *country* documents in this query.)
 22. Complete the join on a key comprised of the literal prefix "*country::*" concatenated with the *address.countryCode* value of the Userprofile document selected through *u*.

23. Select the `name` property of the Country document joined as `c`, as `countryName`.

```
SELECT u.firstName, u.lastName, u.email, c.name AS countryName
FROM couchmusic2 AS u
USE KEYS "userprofile::abandoninghouseclean34190"
JOIN couchmusic2 AS c
ON KEYS "country::" || u.address.countryCode;
```

The screenshot shows the Couchbase Query Editor interface. At the top, there is a 'Query Editor' header with a 'history (67/67)' link. Below it is a code editor containing the provided SQL-like query. The 'countryName' column is highlighted with a red box. Below the code editor are 'Execute' and 'Explain' buttons, and a status message: 'success | elapsed: 2.6ms | execution: 2.64ms | count: 1 | size: 200'. To the right is a 'Preferences' button. Below the code editor is a 'Query Results' section. It has tabs for 'JSON', 'Table', 'Tree', 'Plan', and 'Plan Text'. The 'JSON' tab is selected, showing a single document with a red arrow pointing to the 'countryName' field. The document structure is as follows:

```

1 - [
2 - {
3 -   "countryName": "United Kingdom of Great Britain and Northern Ireland",
4 -   "email": "leo.beck@email.com",
5 -   "firstName": "Leo",
6 -   "lastName": "Beck"
7 - }
8 - ]

```

B. Join documents by prefixing values from an embedded key array

9. In the Couchbase console, open the `couchmusic2` bucket and filter for documents having the term "`playlist::`" as a prefix.
10. Open the first Playlist document, review its structure, and notice it include a `track` array of string values, each representing an (un-prefixed) Track document ID.

The screenshot shows the Couchbase Document Editor. At the top, it says "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe". Below the document content are three buttons: 'Delete', 'Save As...', and 'Save'. The document content is a JSON object with several fields. A red box highlights the `tracks` array, which contains a large list of string values representing Track IDs. The full JSON content is as follows:

```

1 {
2   "updated": "2015-09-11T10:40:01",
3   "name": "Playlist # 5 for Morgan",
4   "created": "2014-11-21T23:03:22",
5   "visibility": "PUBLIC",
6   "tracks": [
7     "89B8A853A3BDB76276B9F52549EF6099920008DC",
8     "535BBD871157C6873814F69DF1ED1B47A743908",
9     "0B358A6A3B31D957A7373D09549B3F8046D112AD",
10    "C9FE05D7BA77FF538D8CA2A95E0733AE3248DBFA",
11    "61E71BC154D057DA0297C50BF270A8783239291",
12    "FF6BC306B6FF006B6D6466161B5ADFAFB4457ADS",
13    "2438822DD350BD07C982A32D2BAD7341D3CFDDC7",
14    "DA7F081047B5452FF2B56F6E28336A54A2363B9B",
15    "34D520CF3CEEB131AFF1AFF00FC8E569E1E846C1",
16    "18DFA3B55EAC51B98B46B6E5E0B9812C281D2F3A",
17    "041EB9B0E8790098922F677A6A629E0B15FDCCCA",
18    "DA1D6746DB102E1121DDC3B3FC1FE795462501F9",
19    "89B8A853A3BDB76276B9F52549EF6099920008DC",
20    "1FDBCABD02D6DC51E0DD058728973759D707370E",
21    "ED3334952F4781016C9C5483E87A250B5FF83FE2",
22   ],
23 }

```

- In the Query Workbench, select the `owner.username` attribute, and the `name` attribute aliased as `trackname`, from the first Playlist document, by its key ("Document ID").
- Because you will be joining a different document type, distinguish the Playlist documents by aliasing the bucket reference as `p` and appending this prefix to `owner` and `name`.

```
SELECT p.owner.username, p.name AS trackname
FROM couchmusic2 AS p
USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe";
```

Query Editor

history (68/68)

```
1 SELECT p.owner.username, p.name AS trackname
2 FROM couchmusic2 AS p
3 USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe";|
```

Execute Explain success | elapsed: 1.92ms | execution: 1.91ms | count: 1 | size: 112 Preferences

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2 {
3   "trackname": "Playlist # 5 for Morgan",
4   "username": "stockadeseffusing18695"
5 }
6 ]
```

- To join Track documents referenced in this Playlist, join a second reference to the `couchmusic2` bucket using the alias `t`, and select the Track `artist` and `title` attributes, using this prefix.
- Complete the join by referencing the `tracks` keys array in Playlist, remembering that each key must have the `"track::"` prefix added. Add a `FOR .. IN .. END` expression to iterate the array values, exposing each as an inline variable named `trackId`.

```
SELECT p.owner.username, p.name AS trackname, t.artist, t.title, t.mp3
FROM couchmusic2 AS p
USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe"
JOIN couchmusic2 AS t
ON KEYS ARRAY "track::" || trackId
FOR trackId IN p.tracks END;
```

Query Results

JSON Table Tree Plan Plan Text

```
1 [
2 {
3   "artist": "King Curtis",
4   "mp3": "https://stream.song365.co/h/470813/King%20Curtis%20-%20Get%20Ready_(song365.cc).mp3",
5   "title": "Get Ready",
6   "trackname": "Playlist # 5 for Morgan",
7   "username": "stockadeseffusing18695"
8 },
9 {
10   "artist": "Vario \"Olinski",
```

C. Restructure selected data as an inline object

15. Modify the prior query so that the individual track details are returned in a distinct JSON object, as a new attribute named "trackdetails", with nested *artist*, *title*, and *url* attributes holding their corresponding values from the query result.

```
SELECT p.owner.username, p.name AS trackname,
       {"artist": t.artist, "title": t.title, "url": t.mp3 } AS trackdetail
  FROM couchmusic2 AS p
    USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe"
JOIN couchmusic2 AS t
  ON KEYS ARRAY "track::" || trackId
 FOR trackId IN p.tracks END;
```

Notice the modified JSON structure, and consider how this technique may be used to reduce structural impedance between an object model in code, and the final structure of JSON documents returned by a N1QL query.

Query Editor

```
1 SELECT p.owner.username, p.name AS trackname,
2       {"artist": t.artist, "title": t.title, "url": t.mp3 } AS trackdetail
3   FROM couchmusic2 AS p
4     USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe"
5   JOIN couchmusic2 AS t
6     ON KEYS ARRAY "track::" || trackId
7   FOR trackId IN p.tracks END;
```

Execute Explain success | elapsed: 4.56ms | execution: 4.54ms | count: 15 | size: 5481 Preferences

Query Results

```
1- [
2- {
3-   "trackdetail": {
4-     "artist": "King Curtis",
5-     "title": "Get Ready",
6-     "url": "https://stream.song365.co/h/470813/King%20Curtis%20-%20Get%20Ready_(song365.cc).mp3"
7-   },
8-   "trackname": "Playlist # 5 for Morgan",
9-   "username": "stockadeseffusing18695"
10- }
```

JSON Table Tree Plan Plan Text



End of Lab