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 *cbimport* or *cbdocloader*
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:

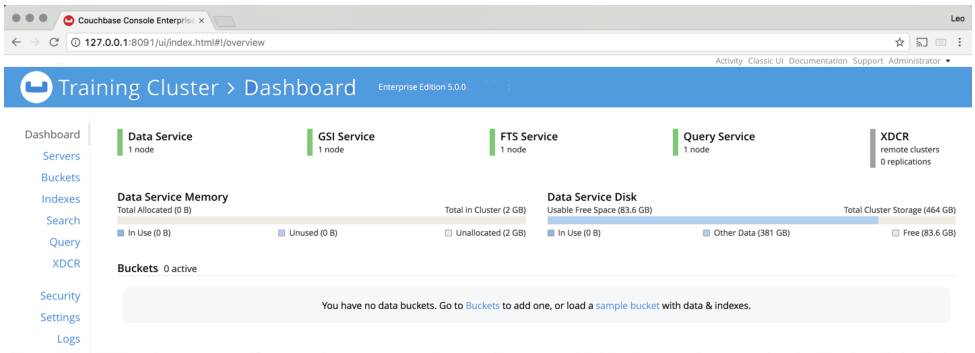
    - ❏ Cluster Name: *Training Cluster*
    - ❏ Admin Username: *Administrator*
    - ❏ Password: *password*
    - ❏ Accept the Terms and Conditions (*check*)
    - ❏ Configure Disk, Memory, Services (*click*)
        - ❏ Data Service (*check*): *2048mb*
        - ❏ Index (GSI) Service (*check*): *1024mb*
        - ❏ 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

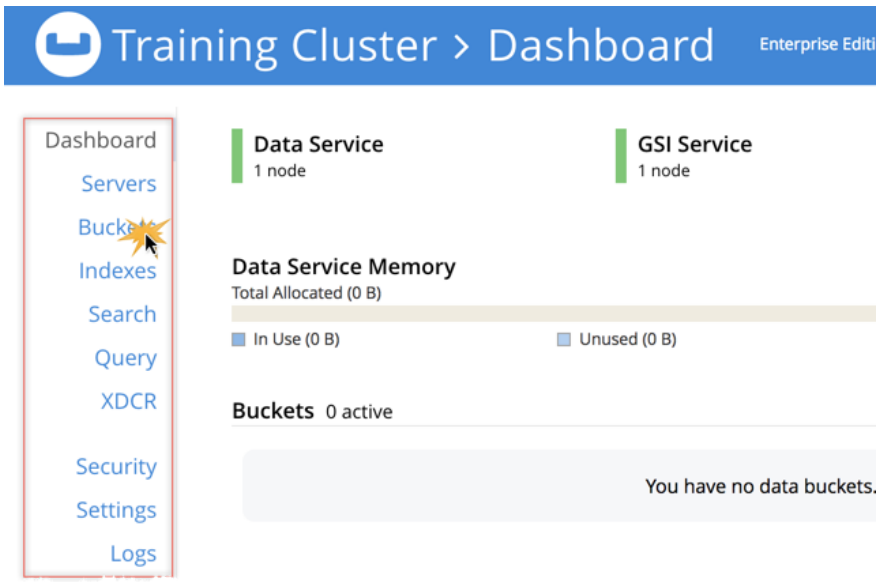
You want a basic orientation to the Couchbase UI.

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

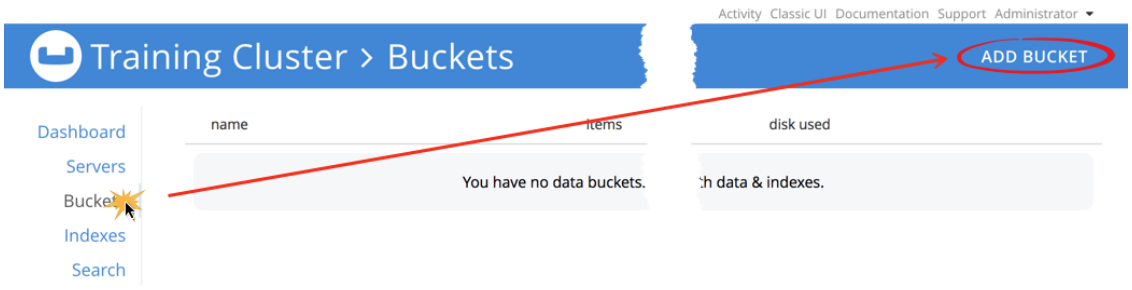
   Documentation is available at http://docs.couchbase.com



C. Create and configure a Couchbase bucket

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

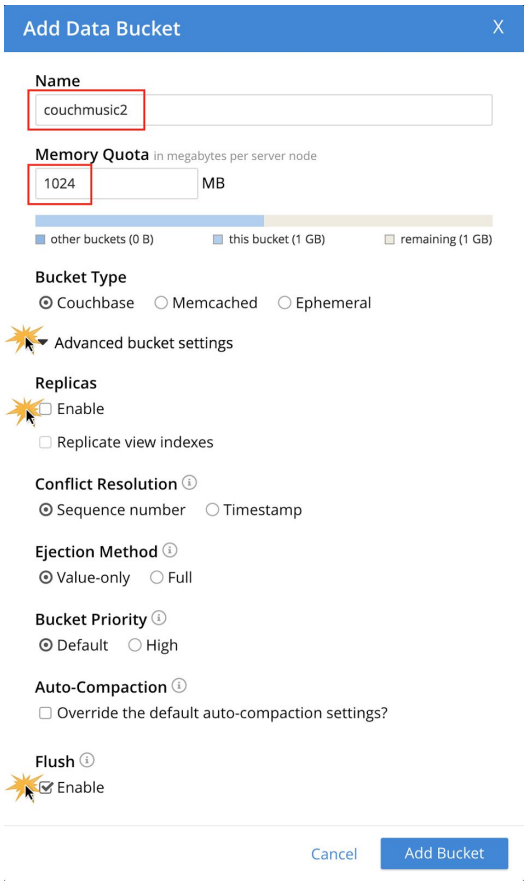
8. In the Couchbase UI, select the *Buckets* panel, and choose *Add Bucket*.



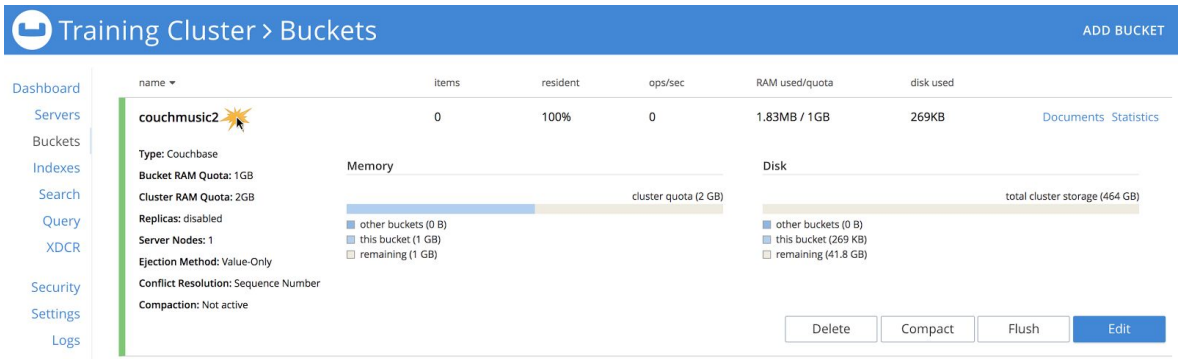
9. In the *Add Data Bucket* panel, set the following values.

    - ❏ 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.



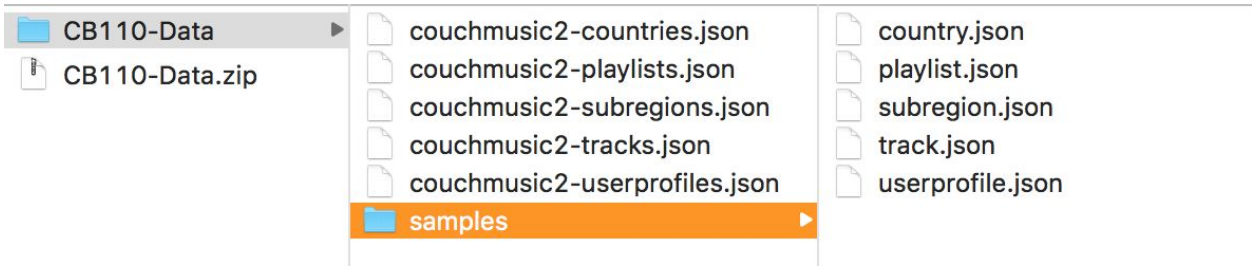
10. Click the bucket name to review summary information for the new *couchmusic2* bucket.



D. Load documents into Couchbase using cbimport

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

11. 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.



12. 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/

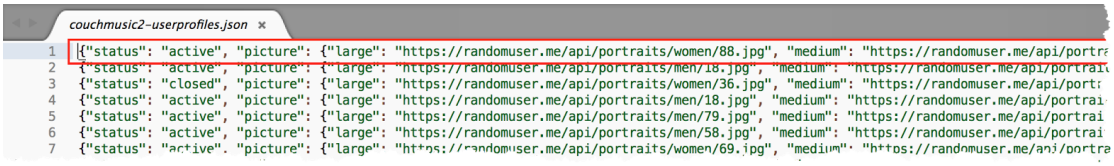
```
couchbase:bin Schuman$ ls
c_rehash                        cbvbucketctl                    gometa
cbbackup                        cbvdiff                         goport
cbbackupmgr                     cbworkloadgen                   gosecrets
cbbackupwrapper                 couch_compact         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

13. 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).



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

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%*

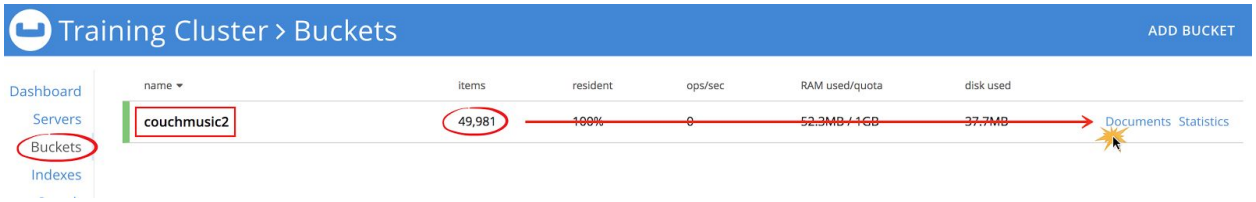
    cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-userprofiles.json -t 2 -g %type%::%username%

    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:

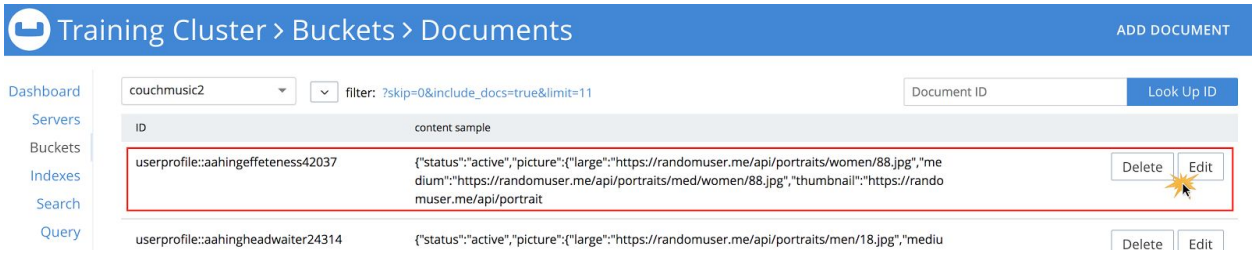
    cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-userprofiles.json -t 2 -g %type%::^%username^%

```
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$
```

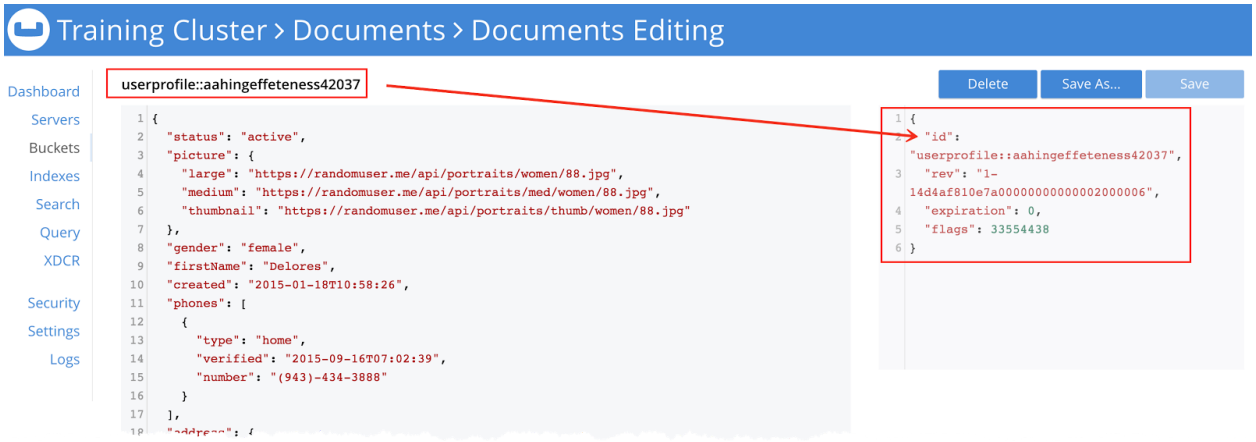
18. In the Couchbase UI, open the *couchmusic2* bucket, examine the number of documents loaded, and open the Documents screen.



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



20. Notice the metadata related to this document.



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**%

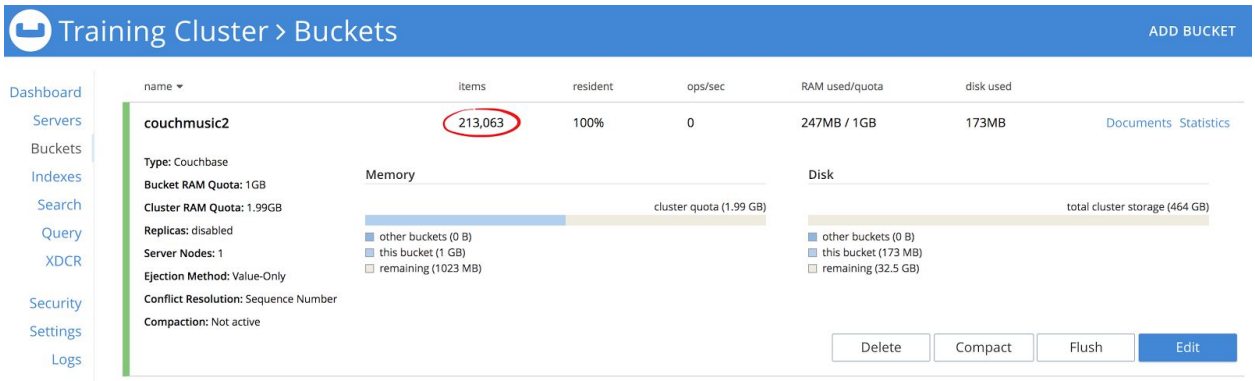
```
cbimport json -c couchbase://127.0.0.1 -u Administrator -p password -b couchmusic2 -f lines -d file://couchmusic2-subregions.json -t 2 -g %type%::%region-number%

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.

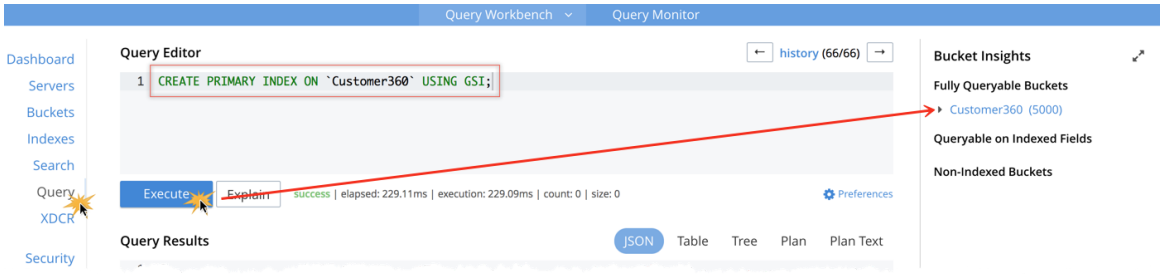


E. Create a primary index to support ad hoc queries (CREATE PRIMARY INDEX)

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;
```



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 <u>not</u> 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

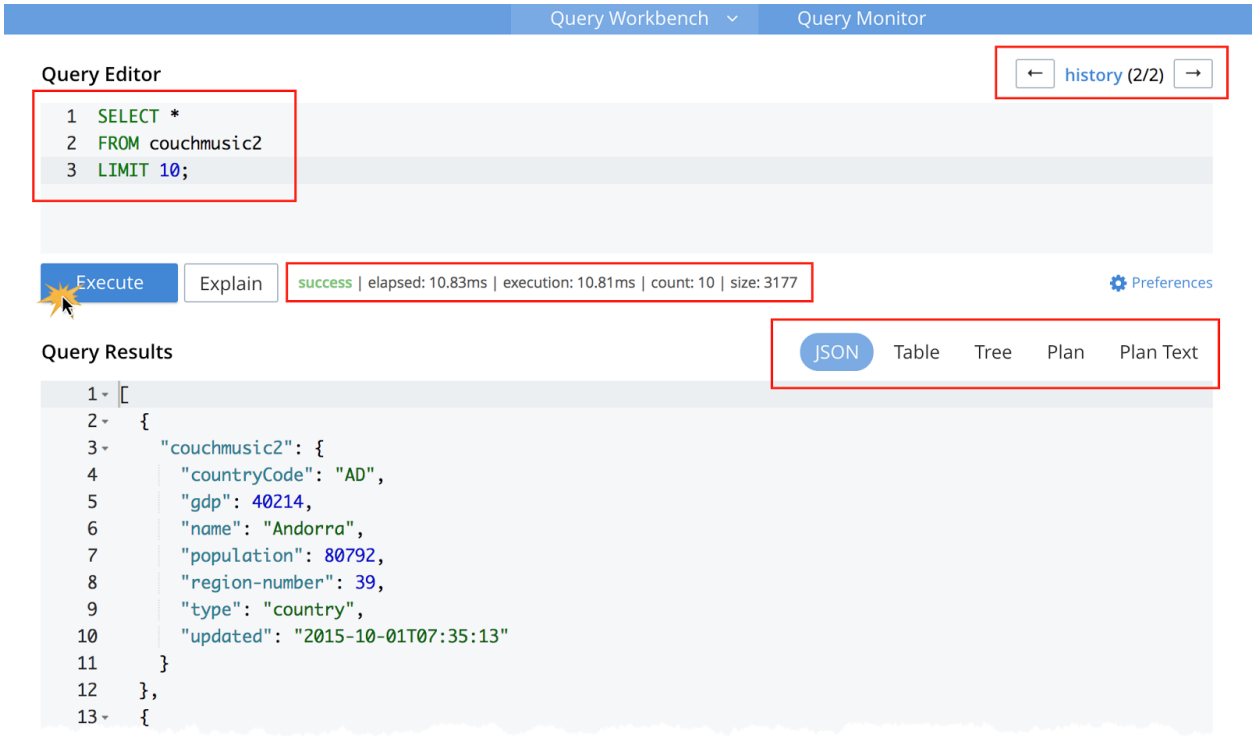
A	Use Query Workbench to select all attributes from all documents, save the results, and limit selection (*, SELECT, LIMIT)
B	Cancel a long-running query
C	(Optional) Execute a query using the *cbq* command line query tool

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

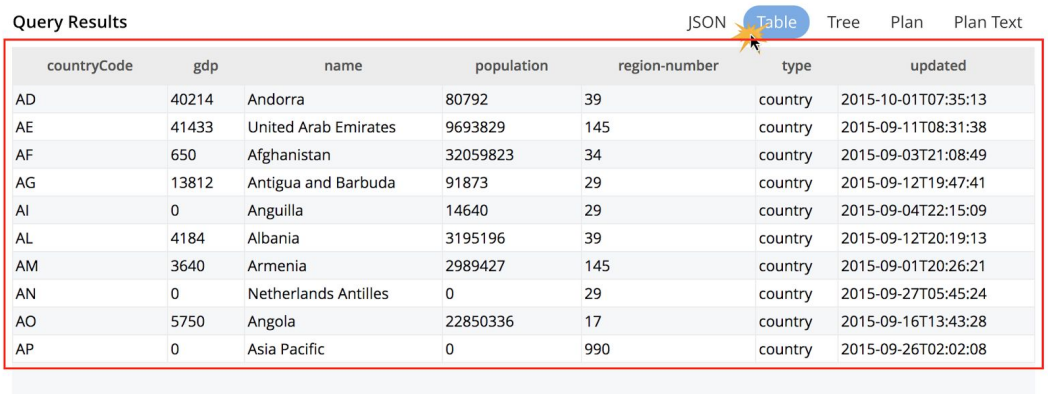
1. In the Couchbase Server UI, select the *Query* tab to open the *Query Workbench*.
2. Write and execute a query to SELECT all attributes from all documents FROM *couchmusic2*, but LIMIT your result to 10 documents.

```
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.

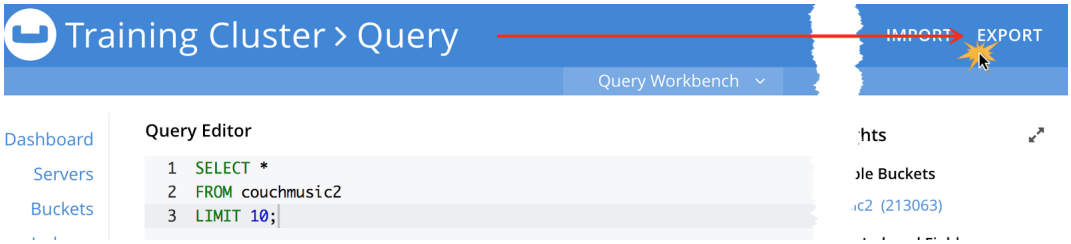


3. Toggle between different displays for the results.

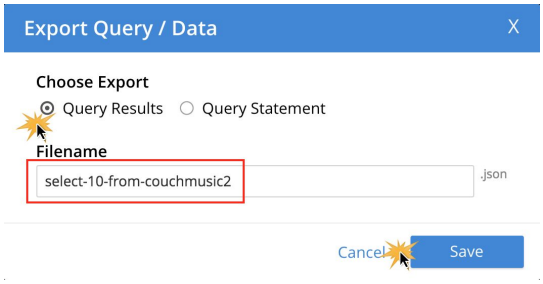


countryCode	gdp	name	population	region-number	type	updated
AD	40214	Andorra	80792	39	country	2015-10-01T07:35:13
AE	41433	United Arab Emirates	9693829	145	country	2015-09-11T08:31:38
AF	650	Afghanistan	32059823	34	country	2015-09-03T21:08:49
AG	13812	Antigua and Barbuda	91873	29	country	2015-09-12T19:47:41
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

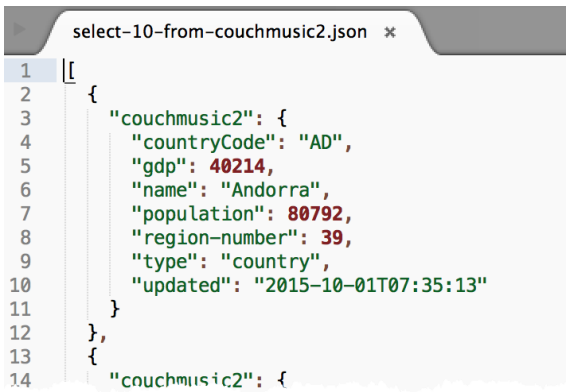
4. Open the query export tool.



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



6. Examine the query results in a text editor.

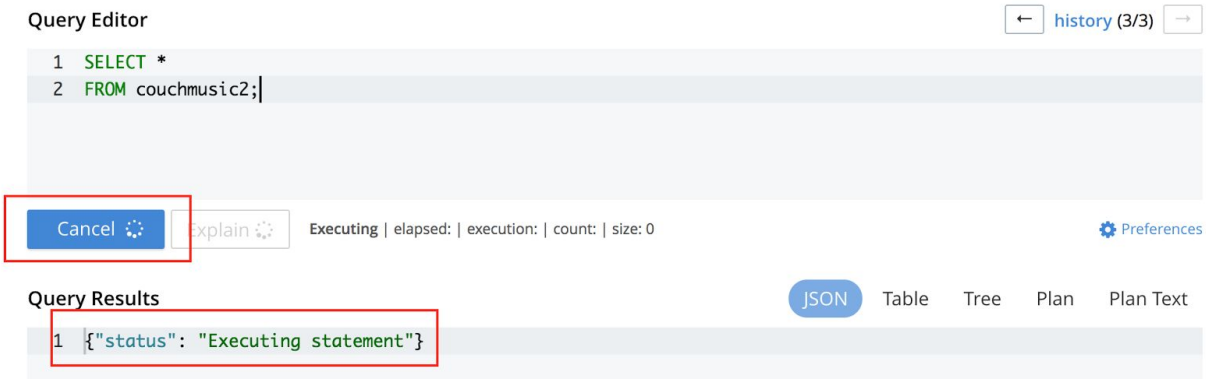


B. Cancel a long-running query

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.



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.
                Default: http://localhost:8091
```

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

End of Lab

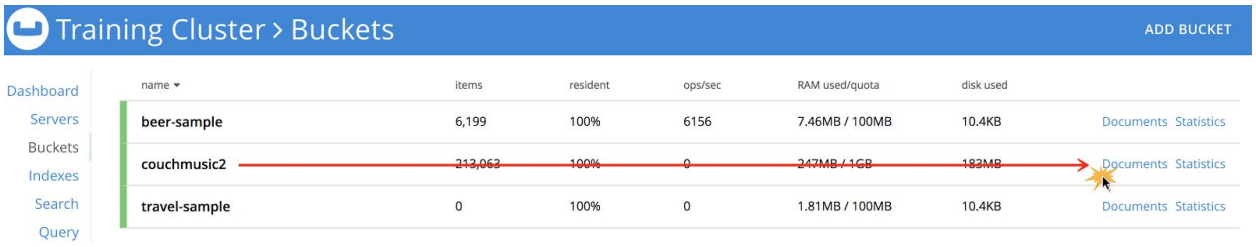
Lab 3 - Selecting nested attributes, aliasing, concatenating, and accessing documents by key

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())

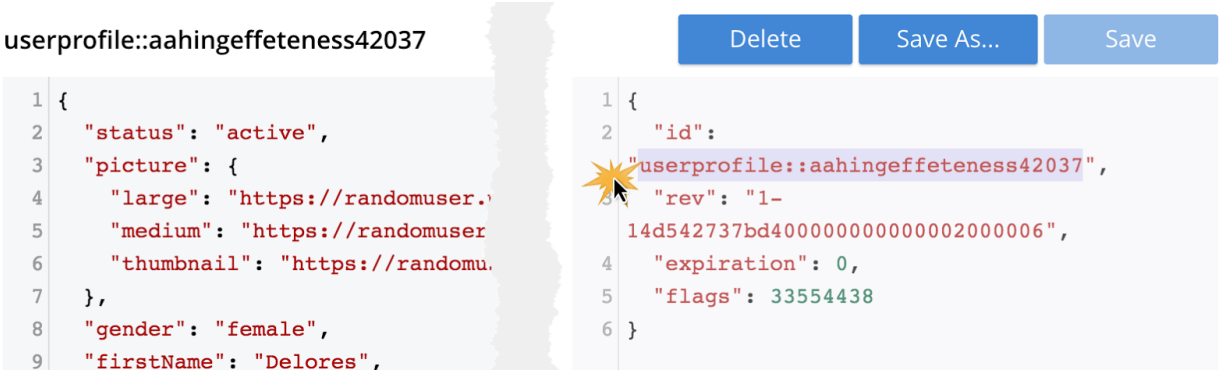
A. Use keys to select specific documents (USE KEYS)

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



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

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



4. 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.

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.

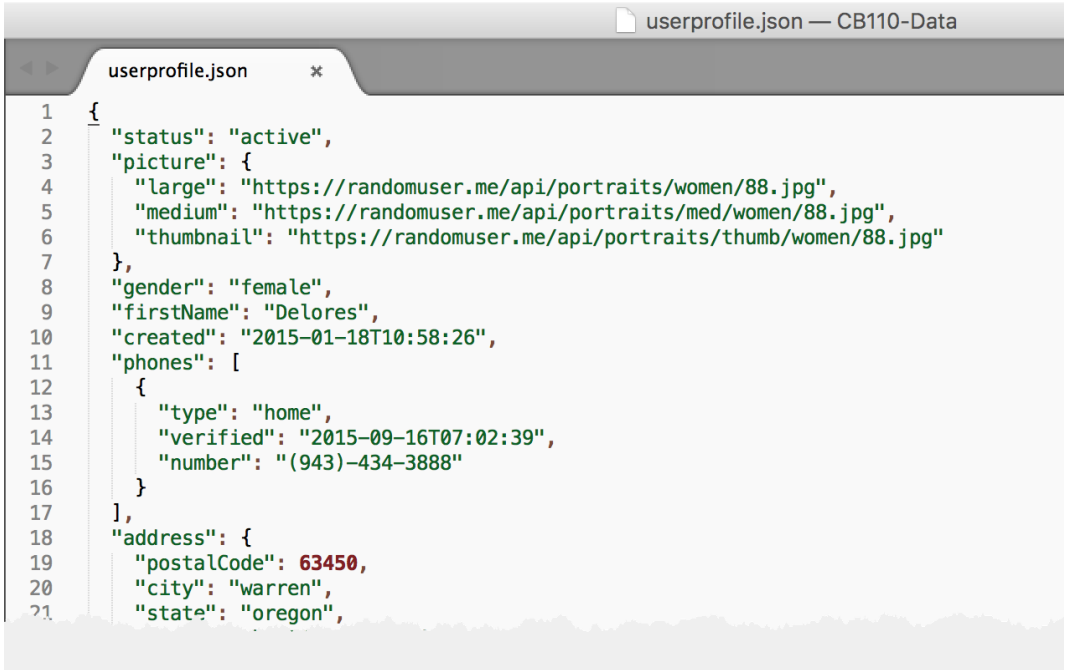
5. Repeat the steps above to copy a second Document ID (aka, "key") from *couchmusic2*.

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"];
```

B. Select named and nested attributes

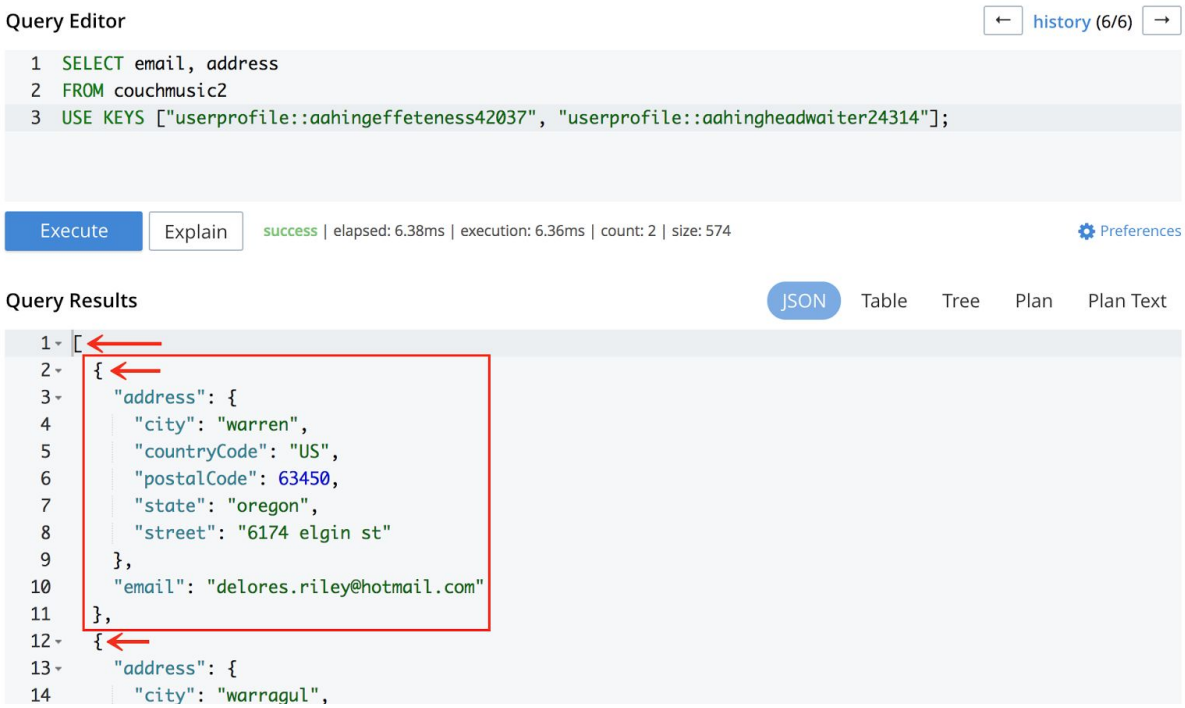
8. From the *CB110-Data/samples* folder, open the *userprofile.json* document. Review the *Userprofile* document attributes and their structure.



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

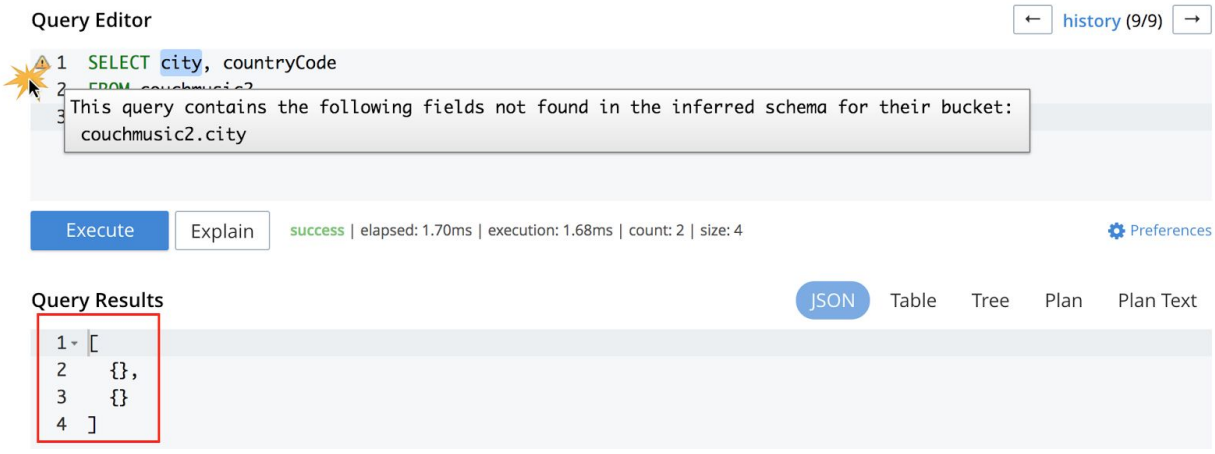
```
SELECT email, address
FROM couchmusic2
USE KEYS ["userprofile::aahingeffeteness42037", "userprofile::aahingheadwaiter24314"];
```

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.



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"];
```



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"];
```

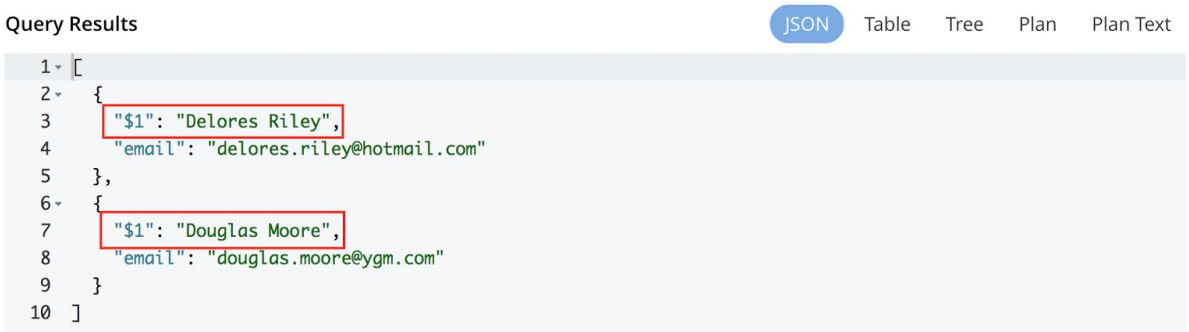
Notice that JSON objects with the selected attributes are returned.

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 *II* 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*.



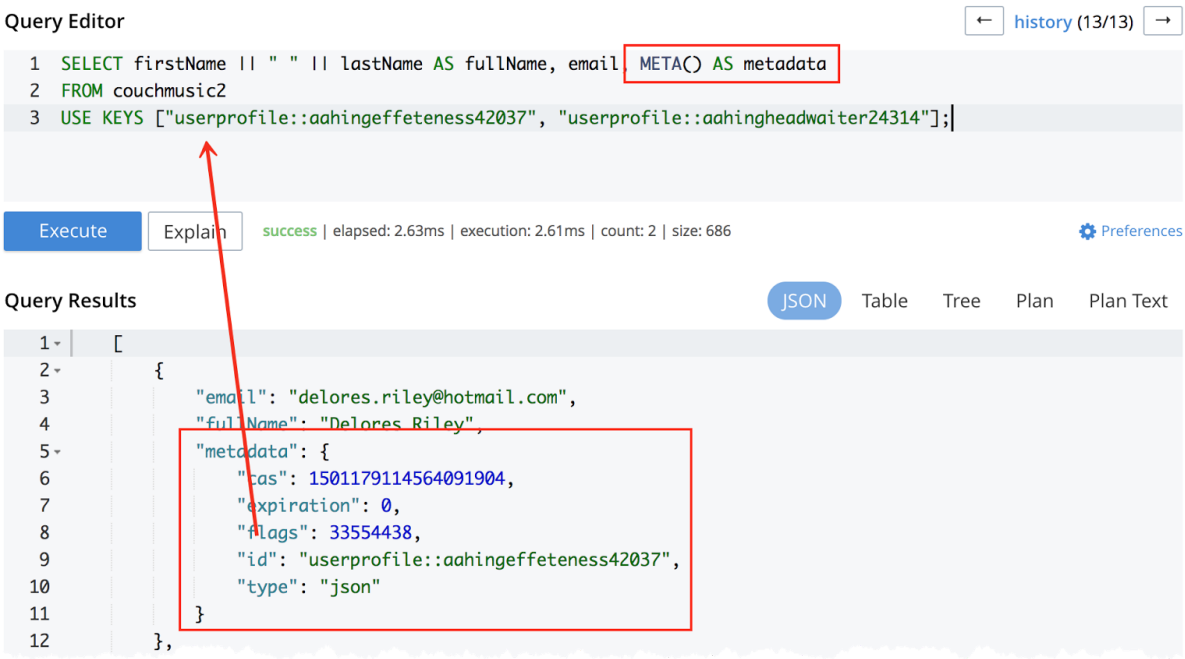
13. Modify the prior query, using AS to alias the concatenated value as *fullName*.

```
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.

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"];
```



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)

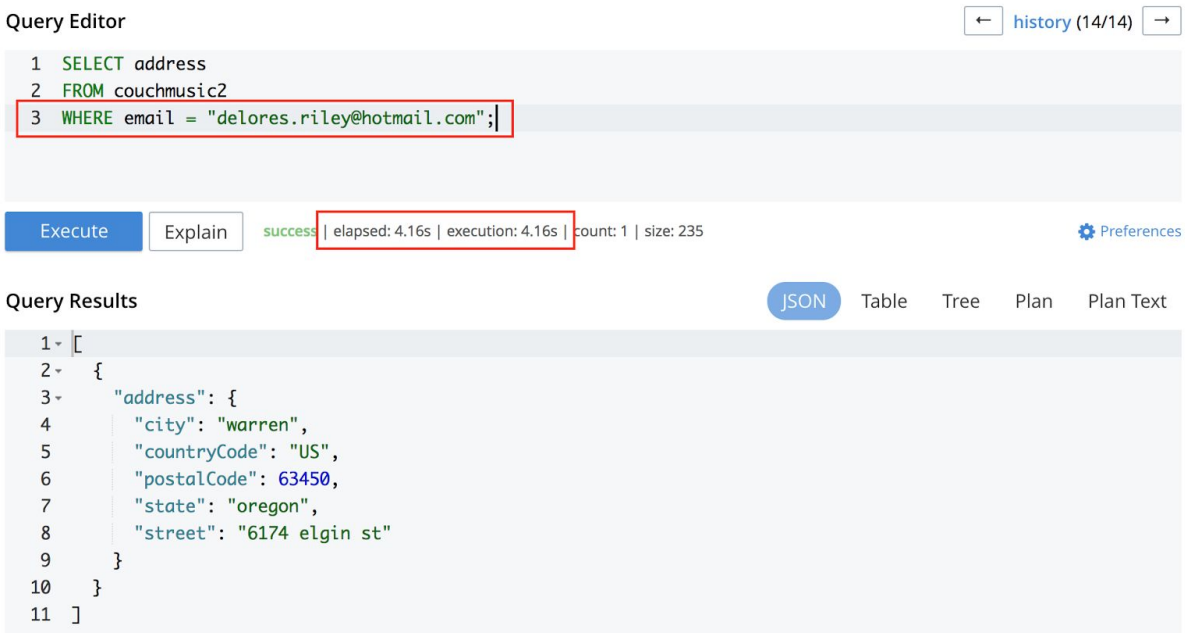
Note, this lab assumes you are already familiar with using the *AND* keyword in SQL, when applying multiple filter clauses to a query.

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.

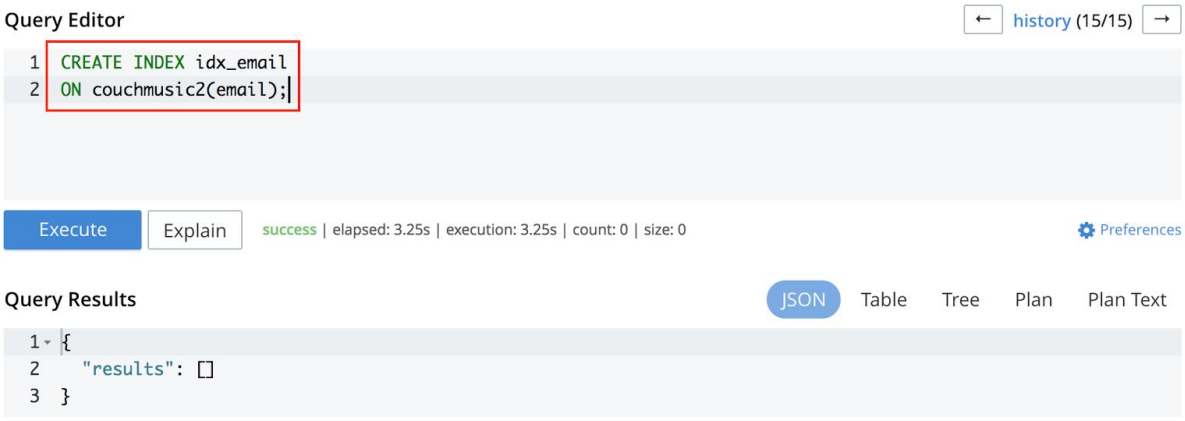


B. Implement a secondary index for a specific document attribute (CREATE INDEX)

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.



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.

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.

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

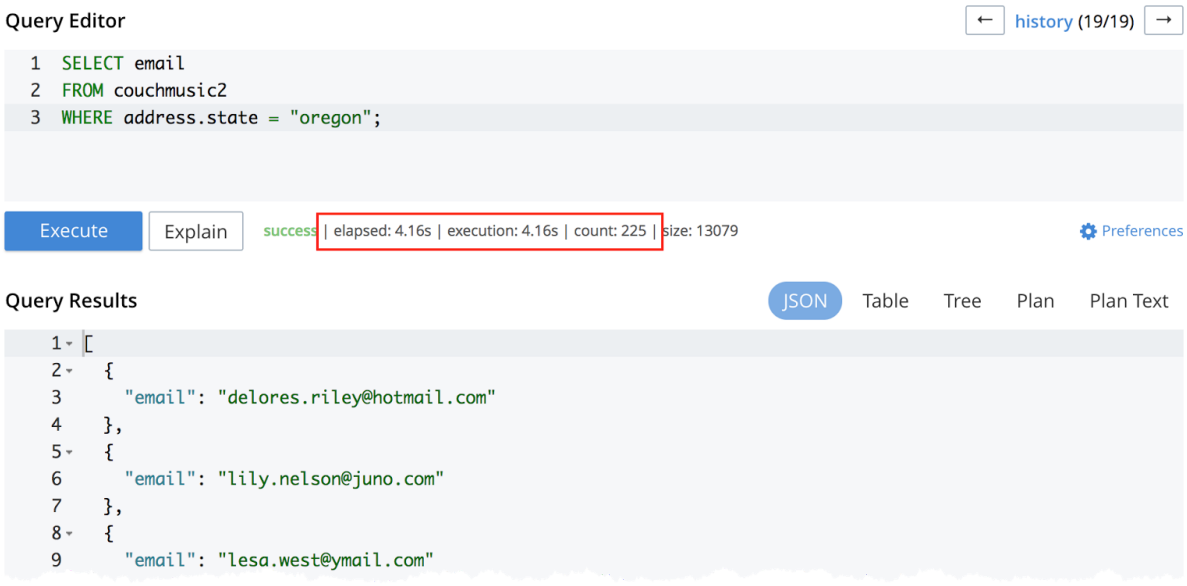
6. Index the *address.state* attributes of *couchmusic2* where the *status* attribute of a document is *"active"*.

```
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.



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.

End of Lab

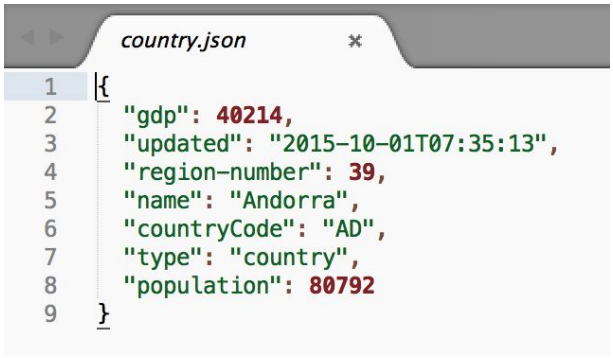
Lab 5 - Querying ranges, ordering results, and explaining query details

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.



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.

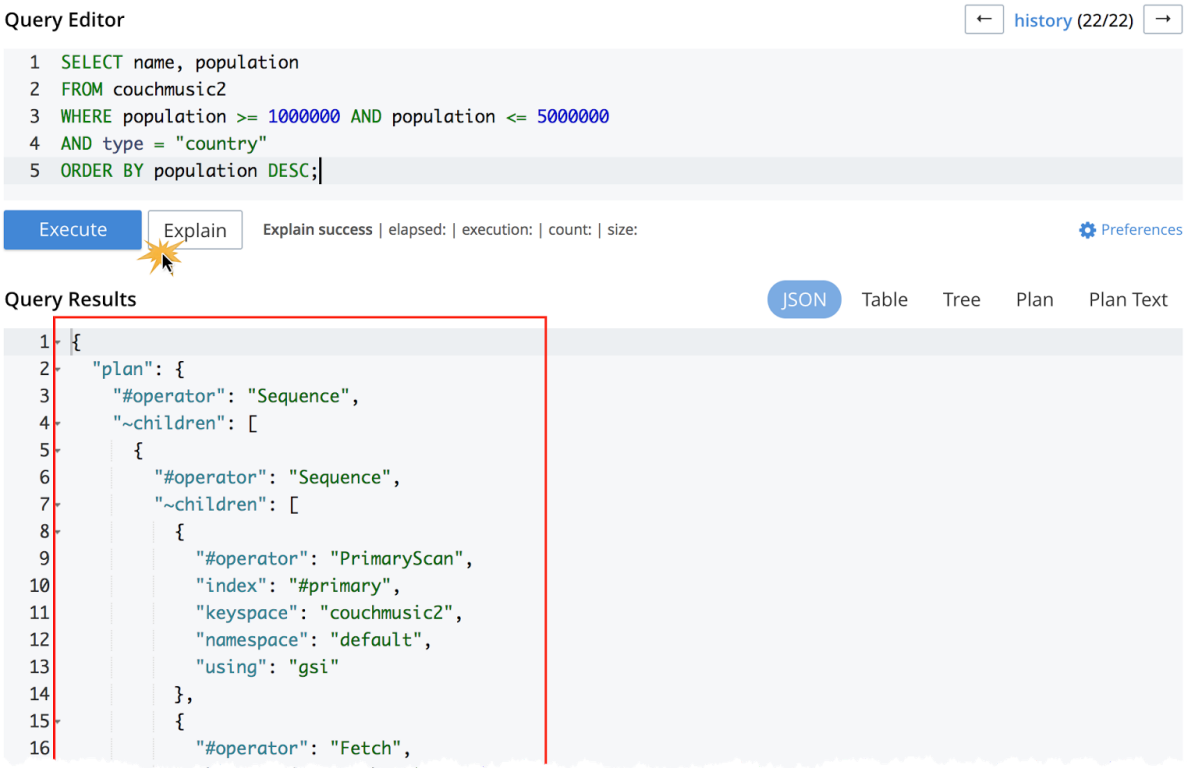
3. Modify the query to sequence the results in descending order, by population.

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

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

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

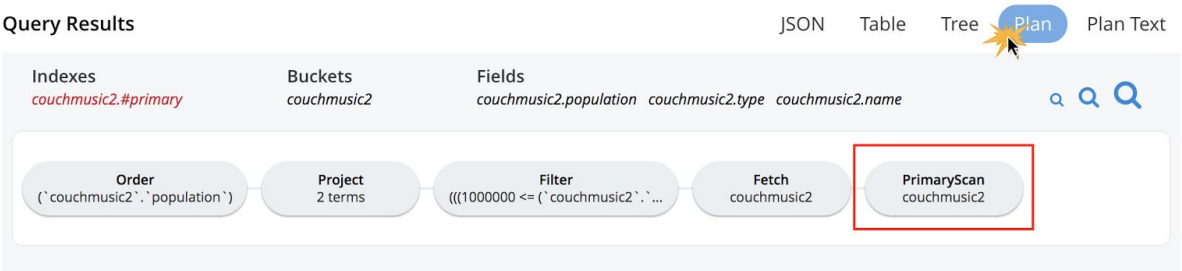
   Note the *#operator* value, which in this case is *PrimaryScan*, indicating that the primary index will be used to perform a full bucket scan to complete this query.



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;
```

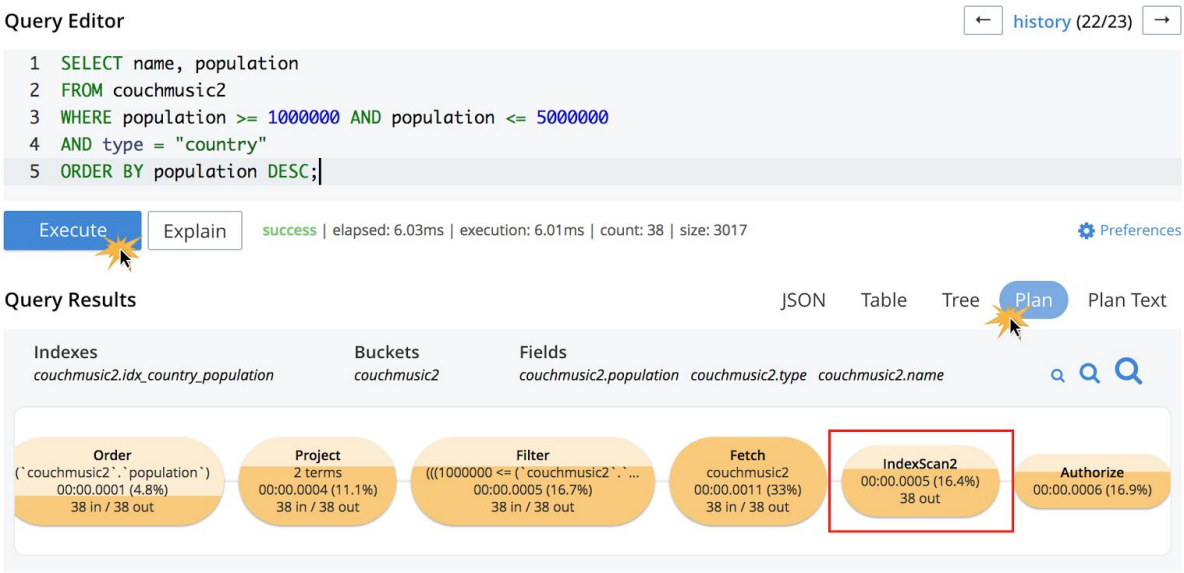
6. Examine the Plan Text and Plan for this query. Notice that the visualized Plan identifies that a PrimaryScan will be used for this query.



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*.

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:

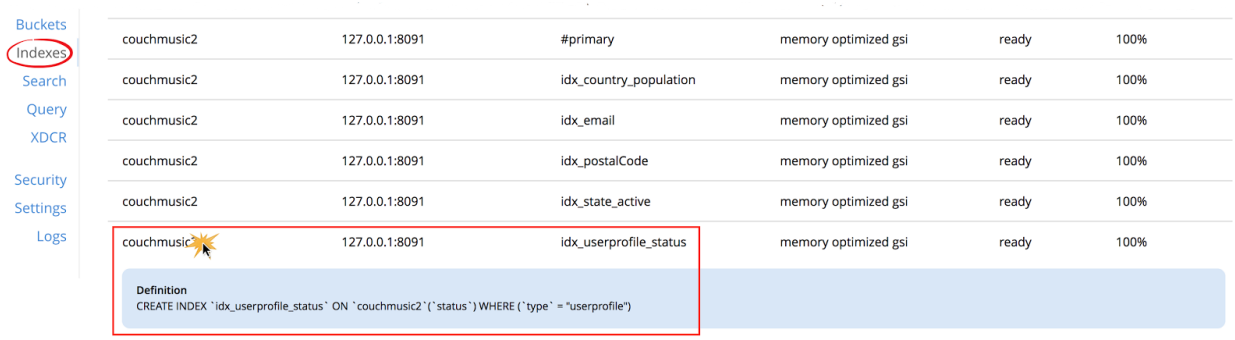
CD210: Couchbase NoSQL Data Modeling, Querying, and Tuning Using N1QL
http://training.couchbase.com

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.



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";
```

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";
```

B. Observe a covering index

4. 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.

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

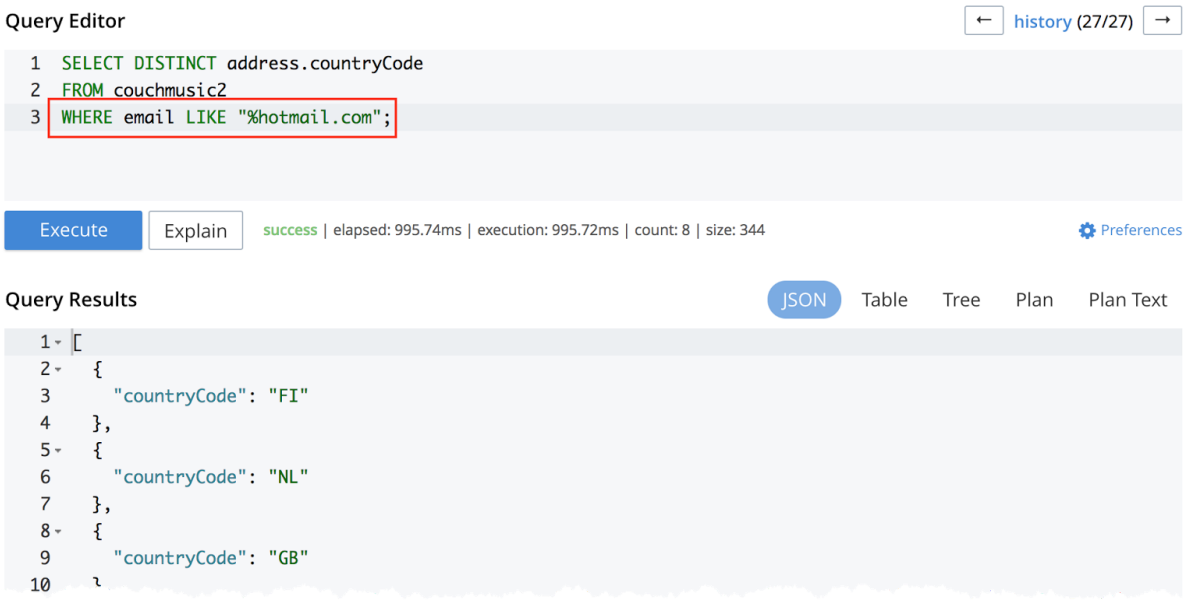
4. In Couchbase console, verify that you created the *idx_email* index in a prior Lab.

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";
```



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;
```

2. 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;
```

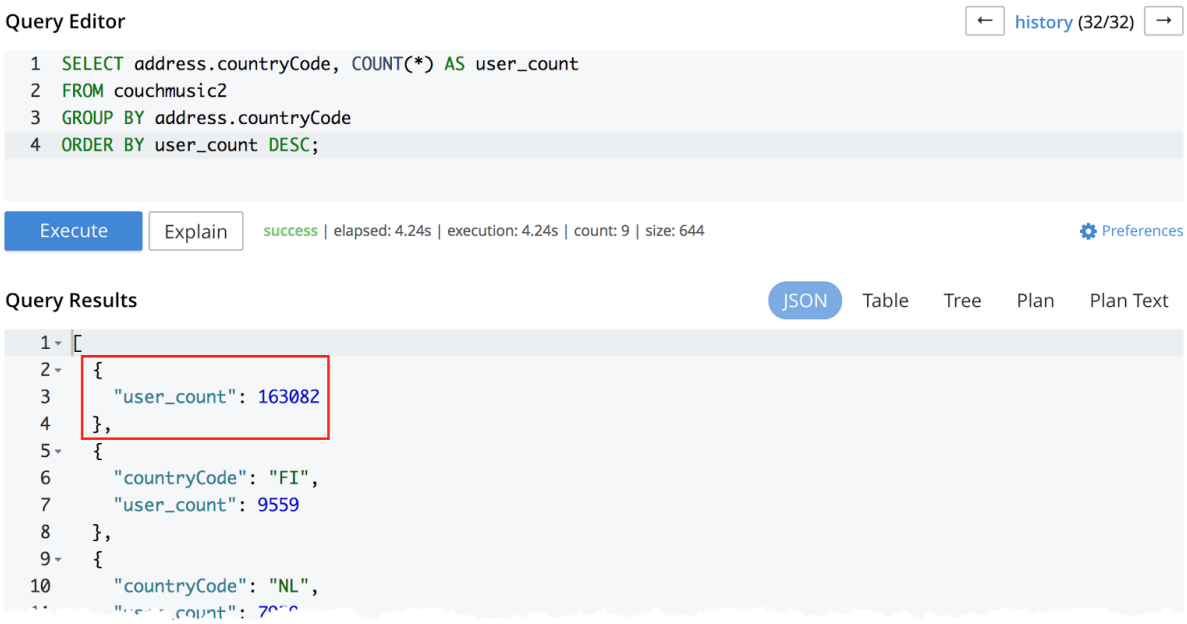
Would an index (aka, a summary table of gender values mapped to corresponding document keys) help determine which *userprofile* documents are missing this value? Or, is a full bucket scan functionally required to determine missing attributes?

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

3. 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.



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

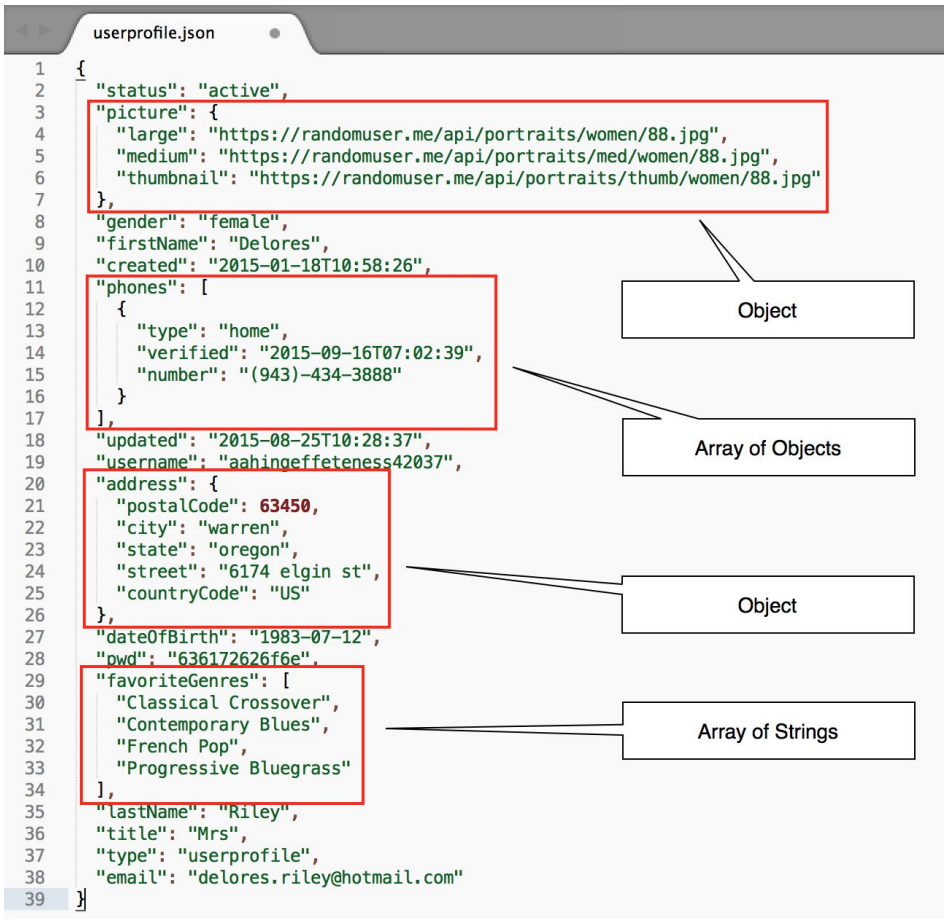
Objectives

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;
```



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.

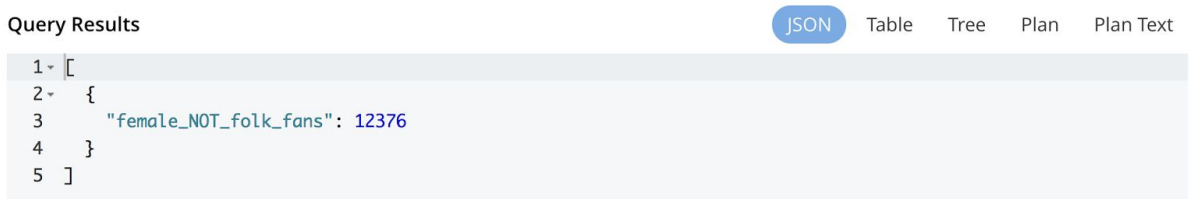
```
CREATE INDEX idx_userprofile_gender
ON couchmusic2(gender)
WHERE type = "userprofile";
```

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";
```

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";
```



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.

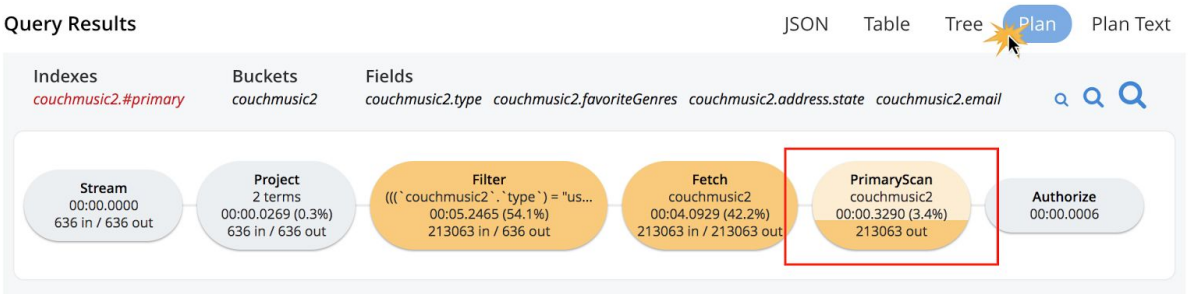
```
CREATE INDEX idx_favoriteGenres
ON couchmusic2 (
  DISTINCT ARRAY genre
  FOR genre IN favoriteGenres END
)
WHERE type = "userprofile";
```

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.

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



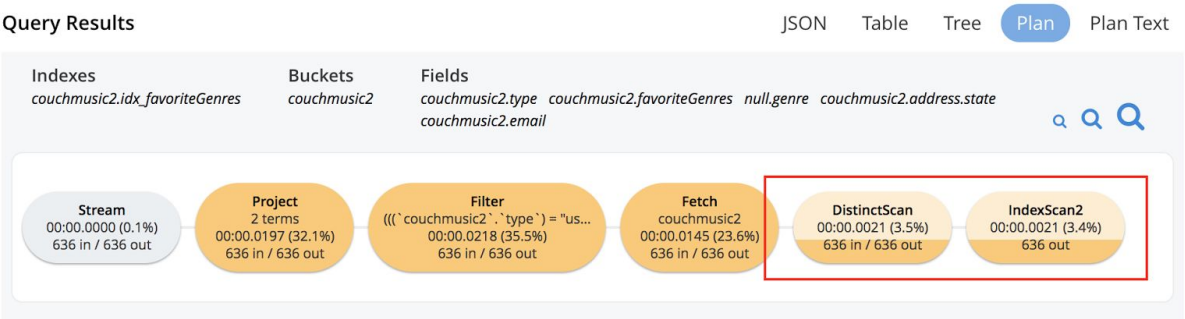
7. Modify the prior query to use expression syntax to select document where ANY *genre* IN *favoriteGenres* SATISFIES the expression *genre* = *"Folk"*.

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

```
AND ANY genre IN favoriteGenres
SATISFIES genre = "Folk" END;
```

Notice the *idx_favoriteGenres* index is now used.

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": "croakerraisiny16166",
      "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;
```

End of Lab

Lab 9 - Using functions and expressions in queries

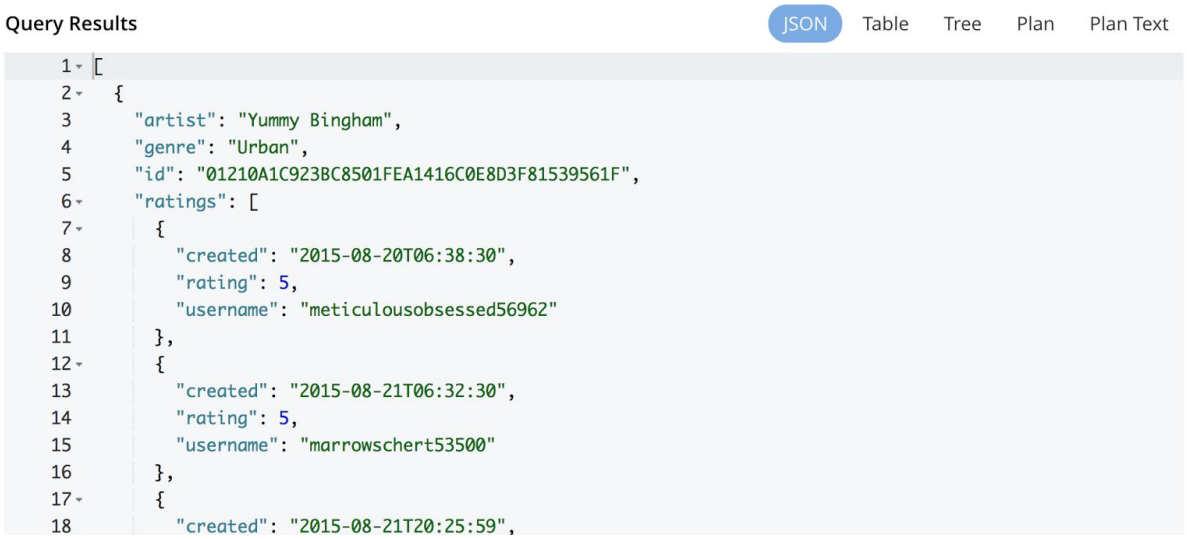
Objectives

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.

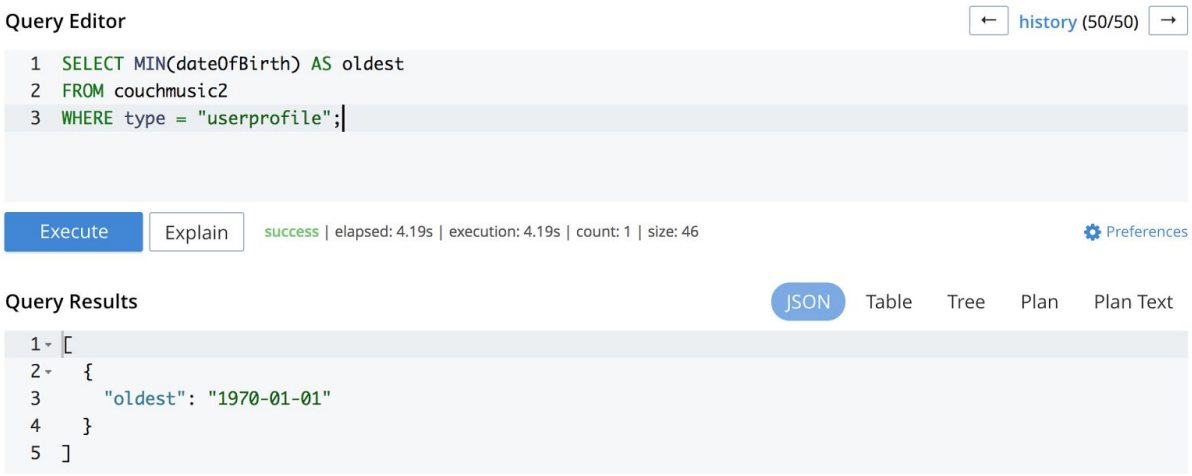
```
SELECT artist, title, genre, id, ratings
FROM couchmusic2
WHERE type = "track"
AND ARRAY_LENGTH(ratings) >= 3
AND EVERY r IN ratings SATISFIES r.rating = 5 END;
```



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

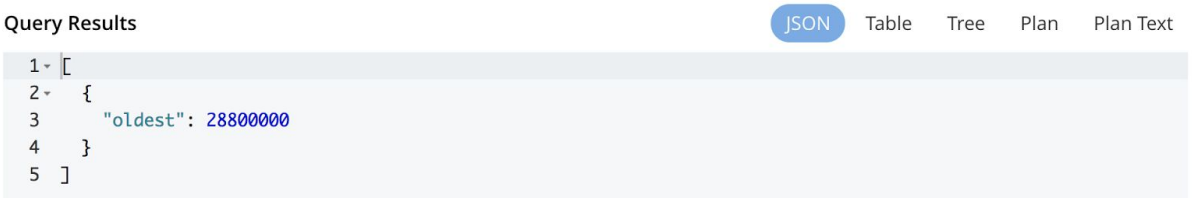
2. Use the MIN() function to select the earliest *userprofile dateOfBirth* as the *oldest* value.

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



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";
```

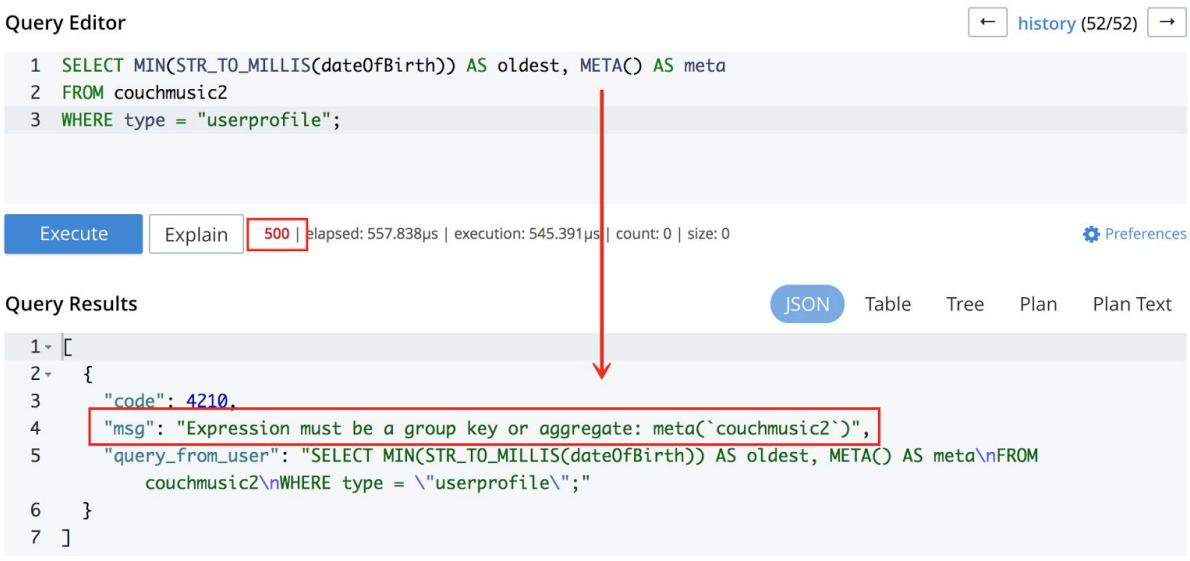


C. Declare inline array to support using functions in aggregate calculations ([…])

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";
```

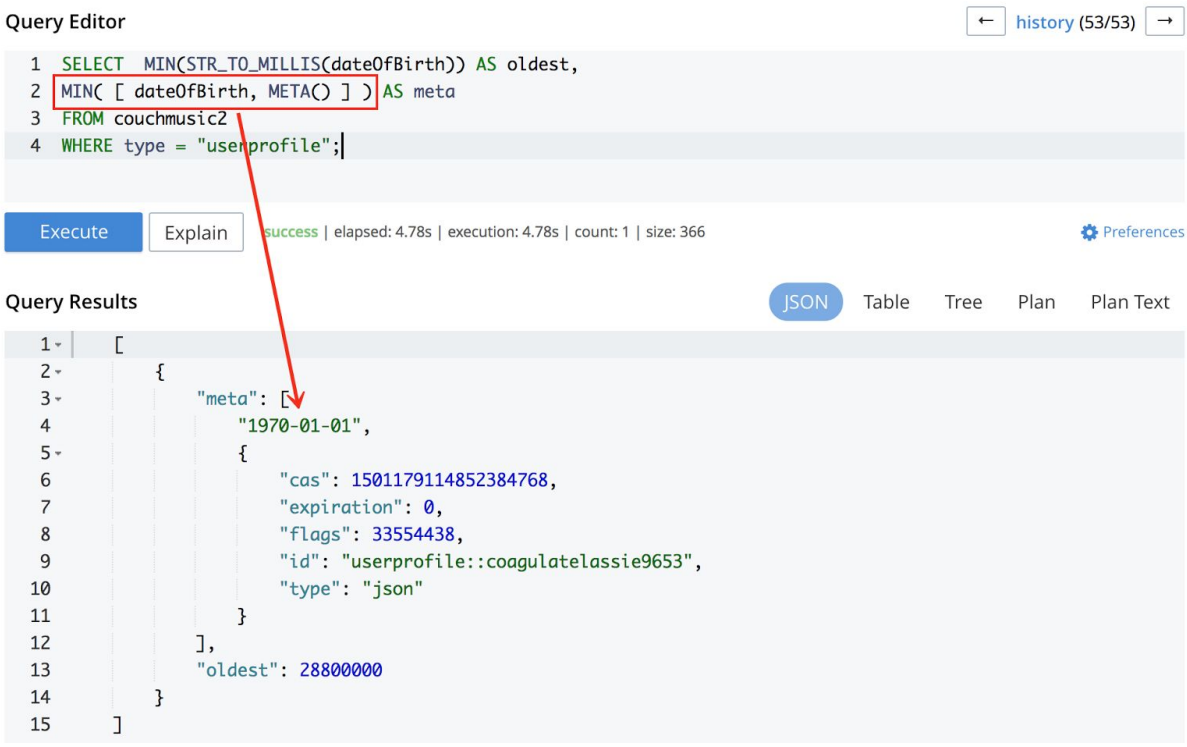
Note, the query will fail. When aggregating, all selected attributes must be aggregated.



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";
```

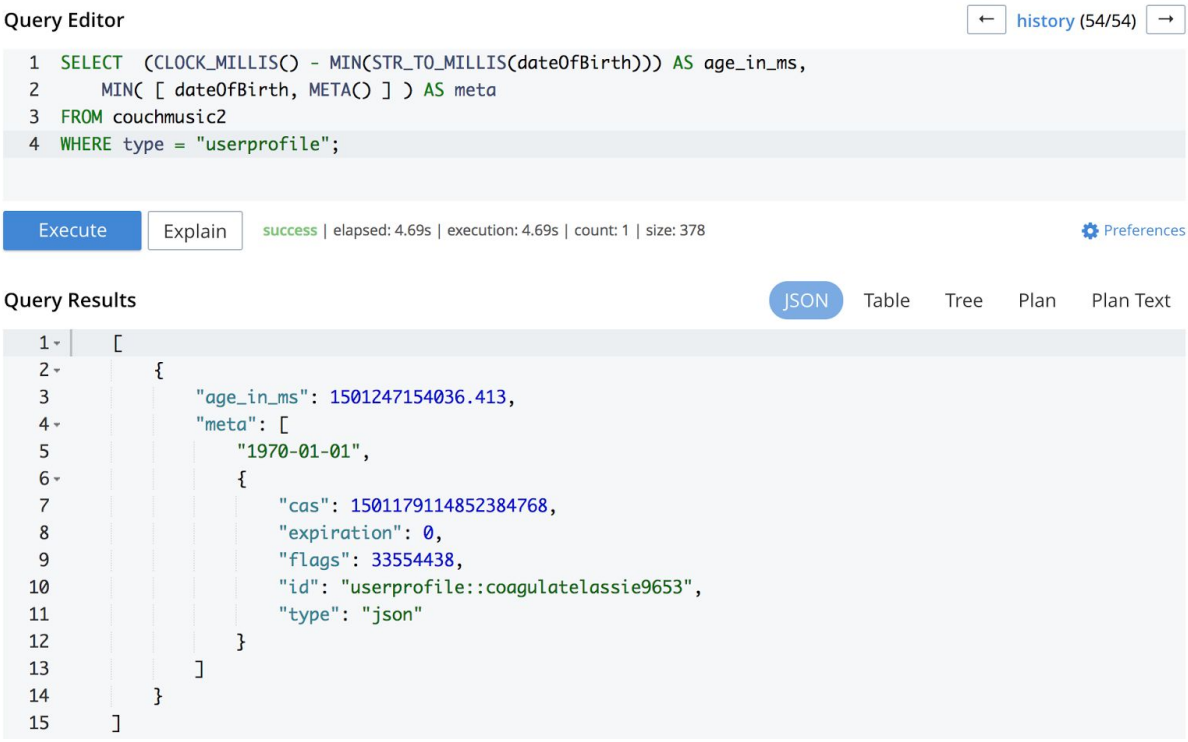
Notice the MIN() function is applied to the first array element: *dateOfBirth*.



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

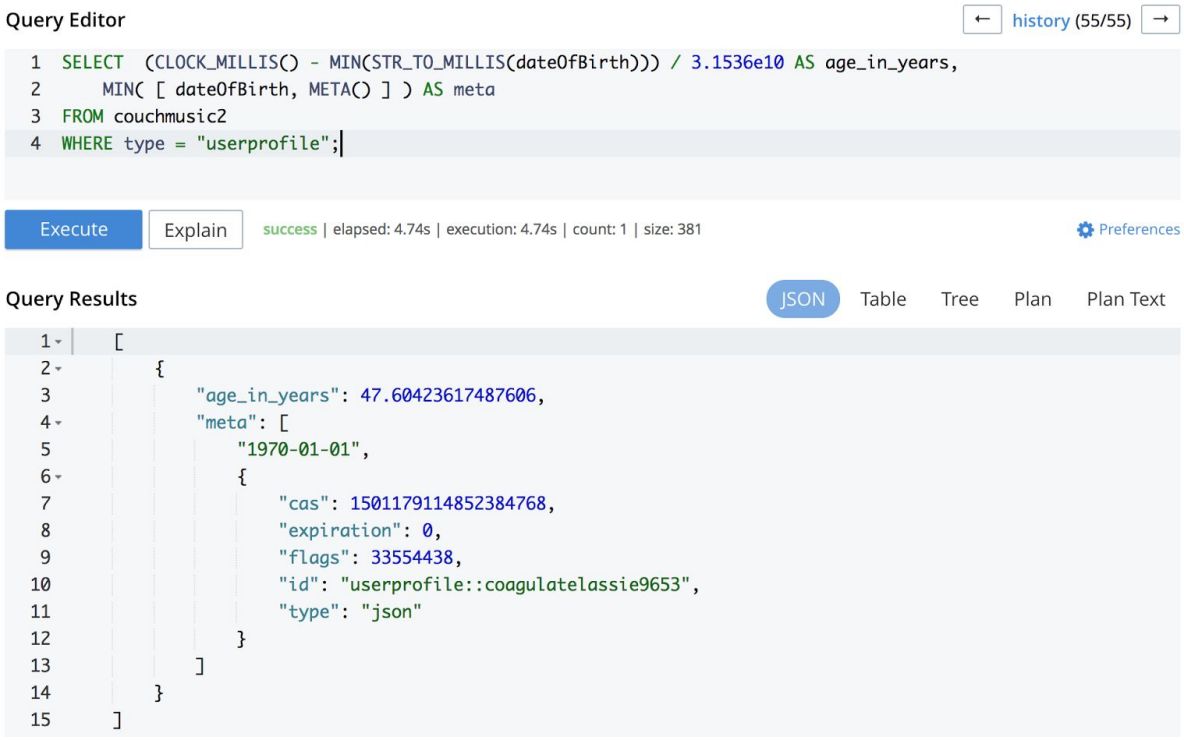
6. Modify the prior query to subtract the birth value, in ms, from the current date, in ms.

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



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";
```



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";
```

End of Lab

Lab 10 - Manipulating data using N1QL DML

Objectives

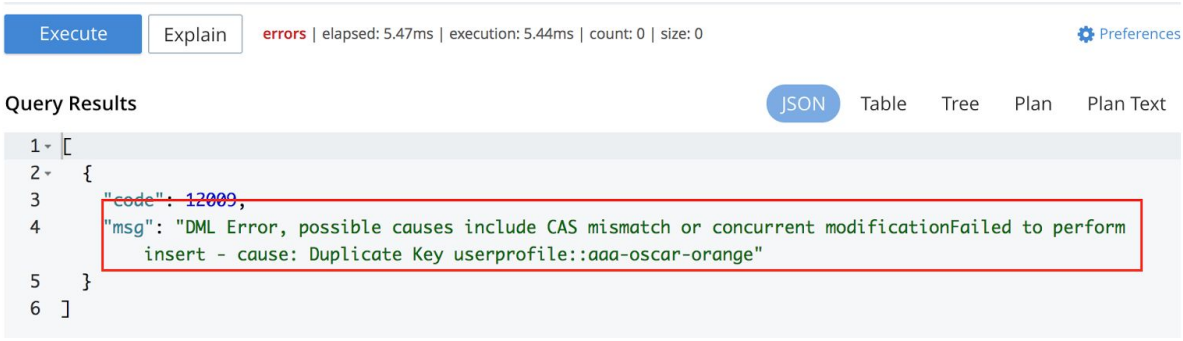
A	INSERT a new document
B	UPSERT and UPDATE new and existing documents
C	DELETE documents by value or key

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"
}
);
```

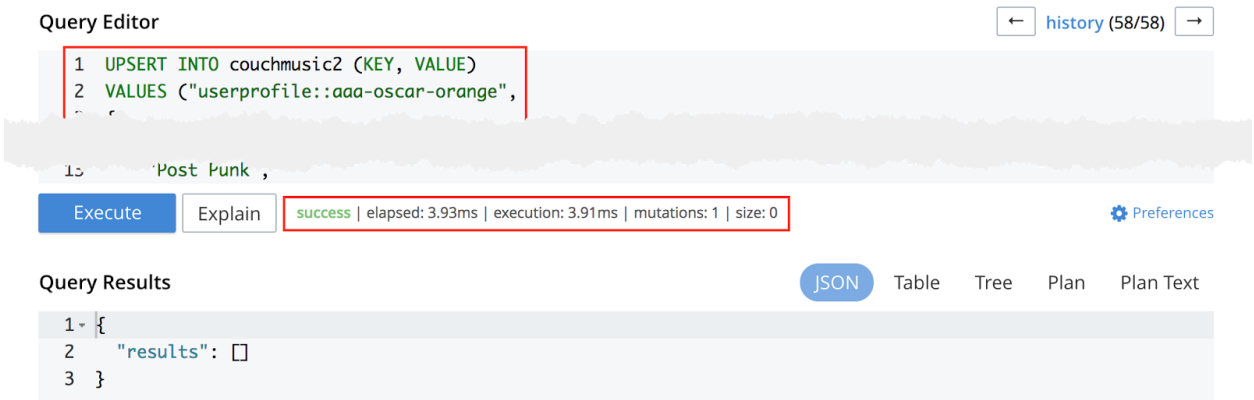
2. Execute this query a second time, and notice the duplicate key error.



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.

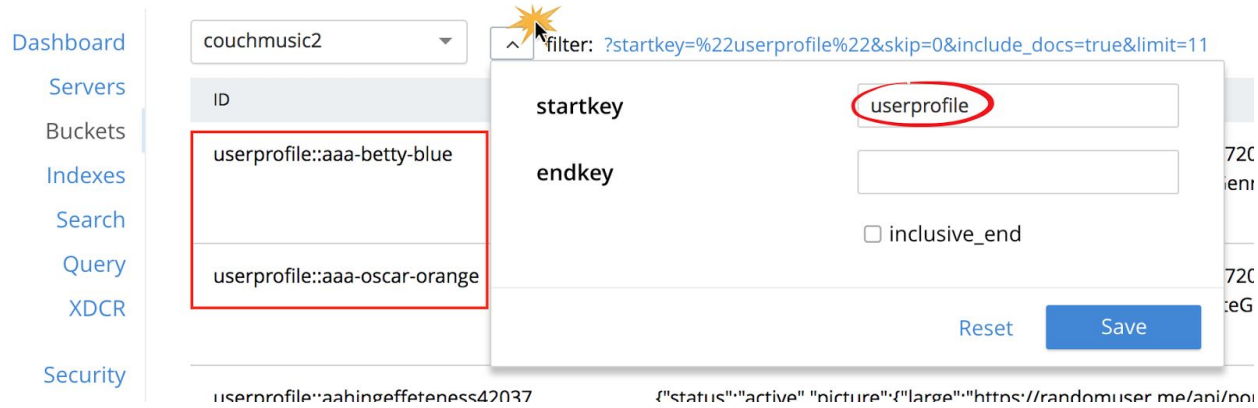
```
UPSERT INTO couchmusic2 (KEY, VALUE)
VALUES ("userprofile::aaa-oscar-orange",
{
  "address": {
    "city": "Portland",
    "countryCode": "US",
    "postalCode": 97203,
    "state": "Oregon"
  },
    ...
}
);
```



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.



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

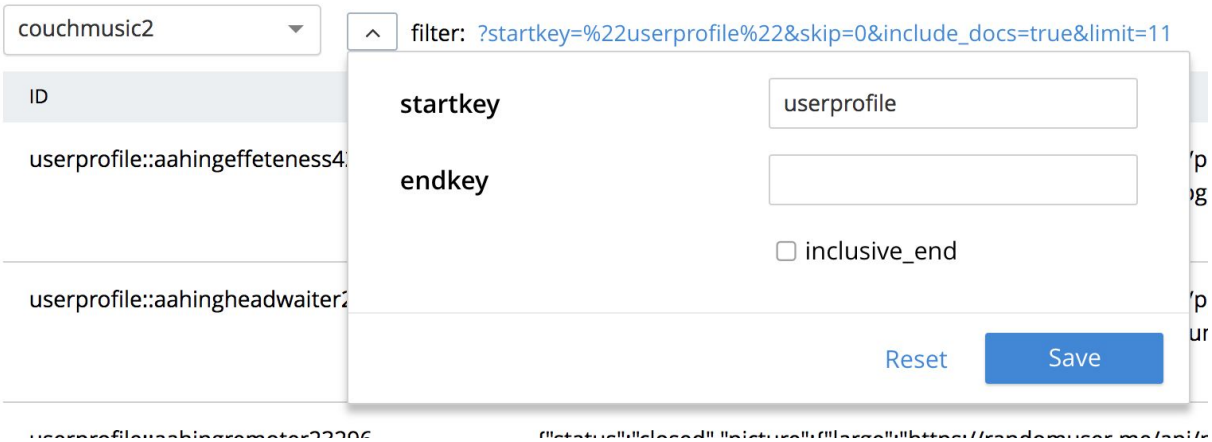
12. Delete *userprofile* documents with this *username*.

```
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.



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

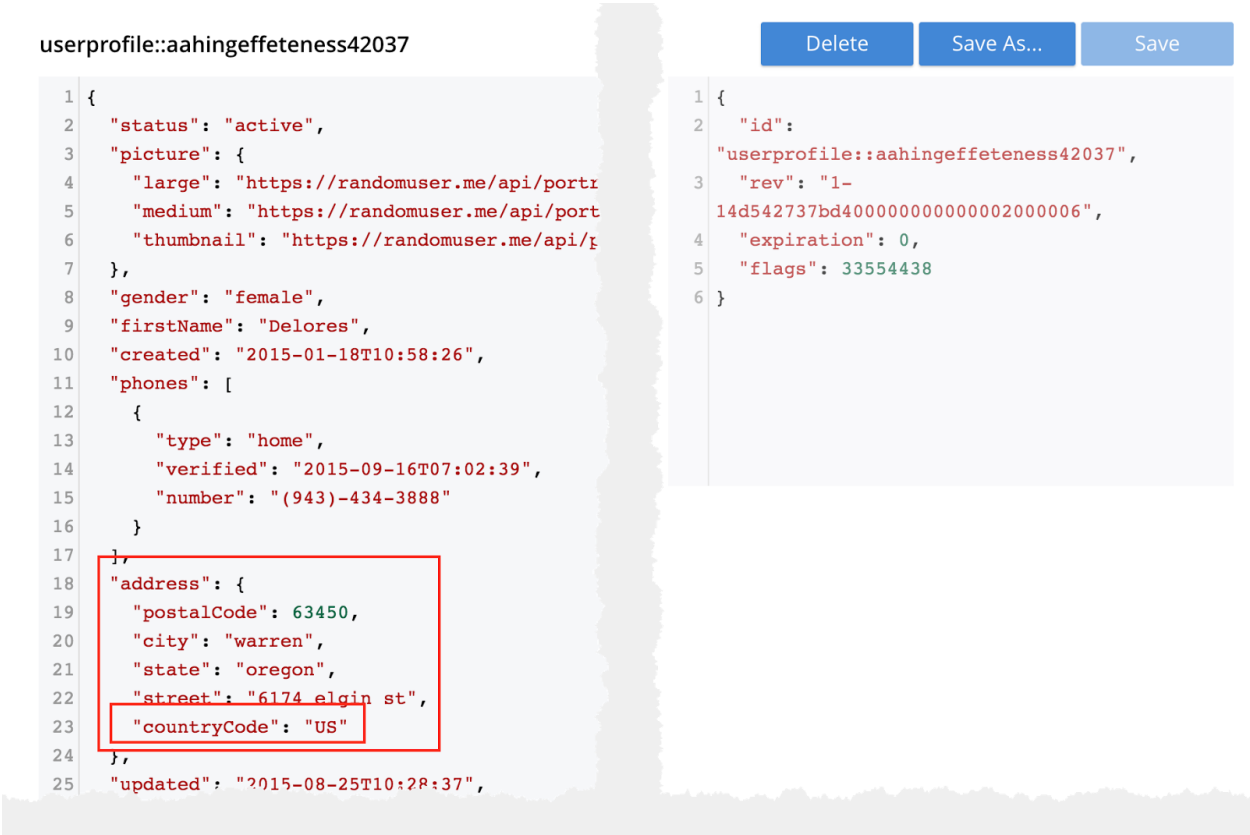
A. Join documents based on an embedded document key

15. Select the *title*, *firstName*, *lastName*, and *email* from *couchmusic2*.
16. Alias *couchmusic* 2 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";
```

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.

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



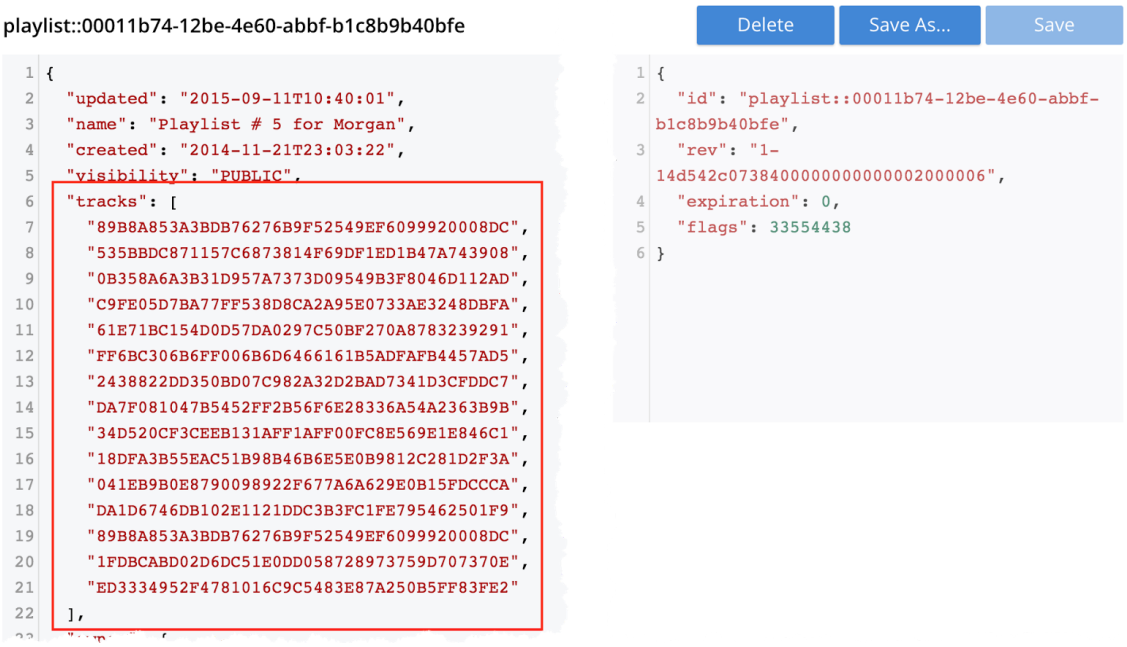
21. Join the *couchmusic2* bucket AS *c* (for references to *country* documents in this query.)
22. Complete the join on a key comprised of the literal prefix *"country::"* concatenated with the *address.countryCode* value of the Userprofile document selected through *u*.

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;
```

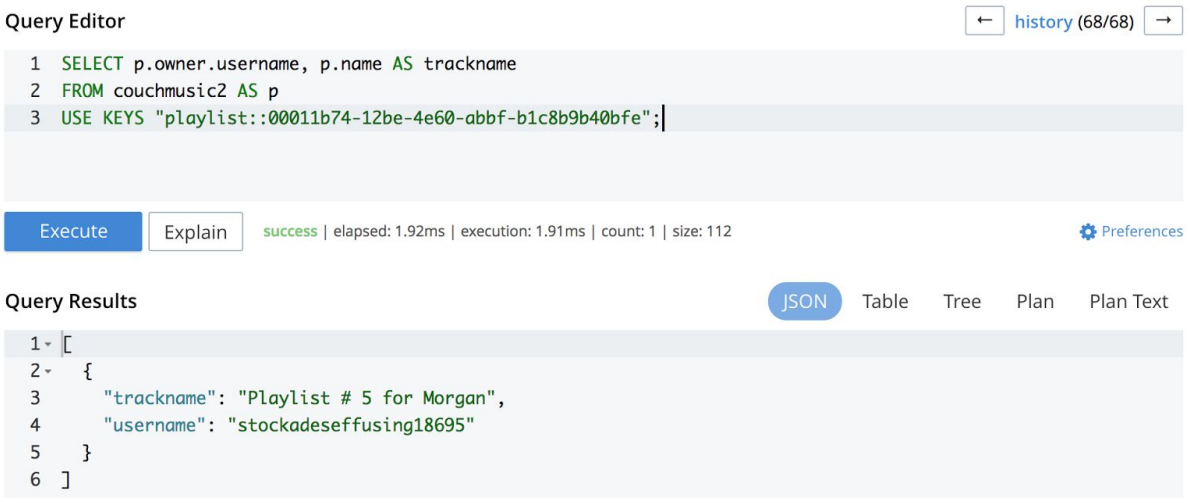
B. Join documents by prefixing values from an embedded key array

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.



11. 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").
12. 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";
```



13. 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.
14. 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*.

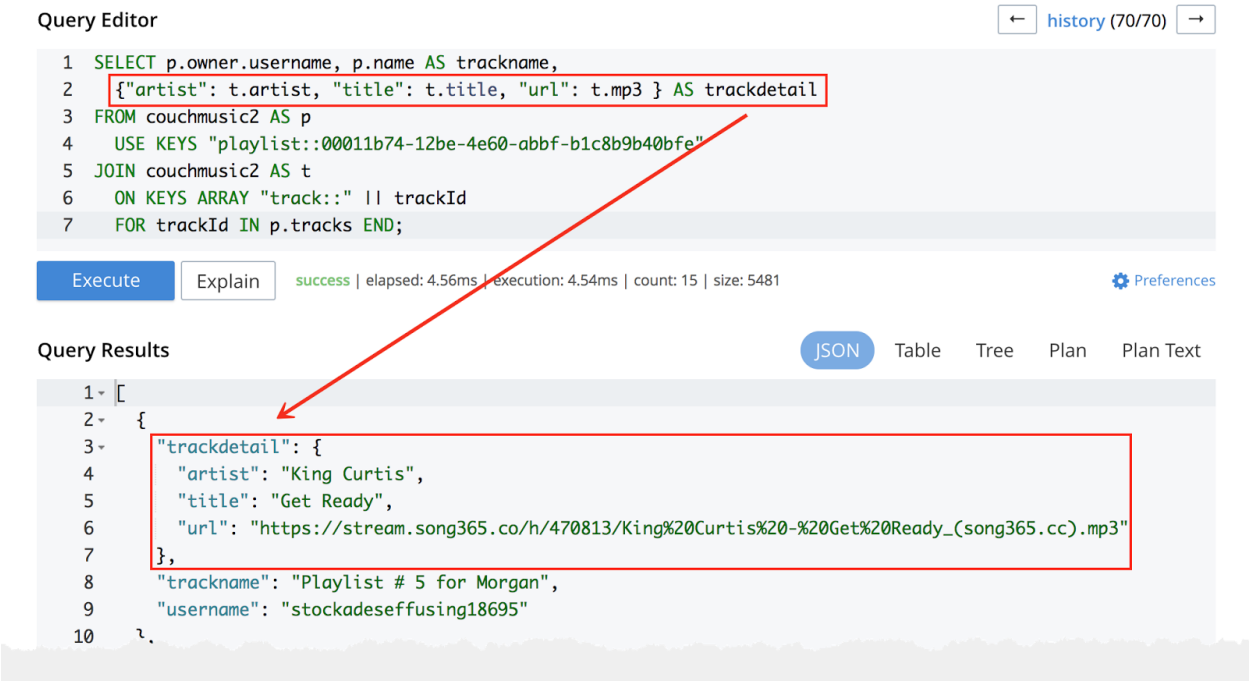
```
SELECT p.owner.username, p.name AS trackname, t.artist, t.title, t.mp3
FROM couchmusic2 AS p
  USE KEYS "playlist::00011b74-12be-4e60-abbf-b1c8b9b40bfe"
JOIN couchmusic2 AS t
  ON KEYS ARRAY "track::" || trackId
  FOR trackId IN p.tracks END;
```

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.



End of Lab