

N1QL & FTS Lab

Lab Objectives



- Identify key parts of the query bench
- Query execution from the terminal
- Understand query performance and explain plans
- Review impact of indexes on query performance
- Exploring various N1QL SELECT features



Query Workbench



BETA BUILD

Activity Documentation Support Administrator ▾

CouchbaseDay > Query

IMPORT EXPORT

Dashboard

Servers

Buckets

Indexes

Search

Query

XDCR

Security

Settings

Logs

Query Editor

1

Execute

Explain

Query Results

1

{"no_data_yet": "Hit execute to run query"}

Query History

X

filter queries



history (5/5)



- 1 Create index idx_city_replicated on `travel-sample`(`city`) with {"num_replica":2}
- 2 Create index idx_city_replicated on `travel-sample`(`city`) with {"num_replica":1}
- 3 Create index idx_city_replicated on `travel-sample`(`city`) with {"num_replica":2}
- 4 Create index idx_city_replicatedx on `travel-sample`(`city`) with {"num_replica":2}

5

Preferences

Delete Selected

Delete All

Plan

Plan Text

Bucket Insights



Fully Queryable Buckets

► [travel-sample \(31591\)](#)

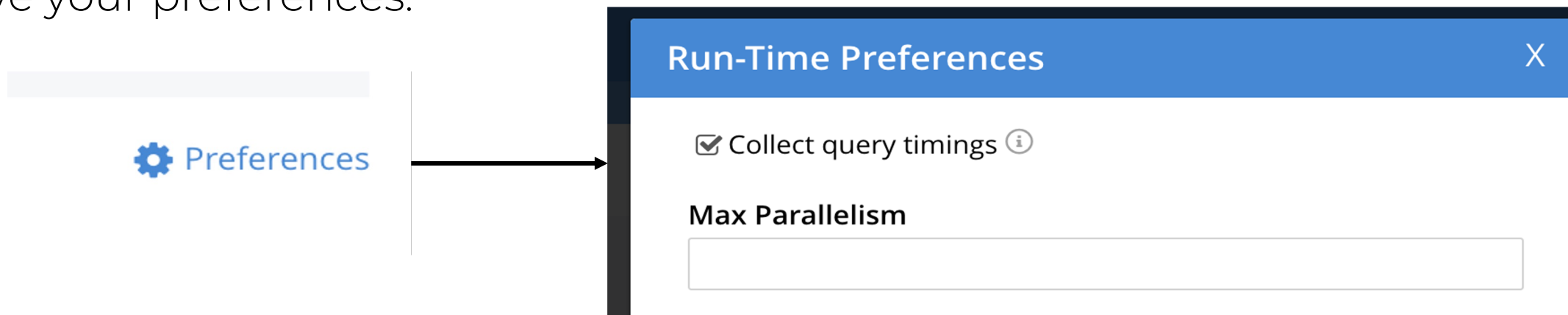
Queryable on Indexed Fields

Non-Indexed Buckets

Exercise – Run-Time preferences



1. Open the Run-Time Preferences window, select “Collect Query Timings” and save your preferences.



1. Open the query workbench and execute the following statement:

```
SELECT *  
FROM `travel-sample` t  
WHERE type = "hotel"  
AND country = "United Kingdom"  
AND ARRAY_LENGTH(public_likes) > 3  
ORDER BY ARRAY_LENGTH(public_likes), ratings DESC
```

Exercise – Query visual plan



1. Navigate to visual query plan
2. What is the execution time ?
3. What indexes are used ?
4. Where does the query spend most of the time ?

[Execute](#) [Explain](#) **success** | elapsed: 496.57ms | **execution: 496.52ms** | count: 238 | size: 2127663 [Preferences](#)

Query Results

[JSON](#)

[Table](#)

[Tree](#)

[Plan](#)

[Plan Text](#)

Indexes

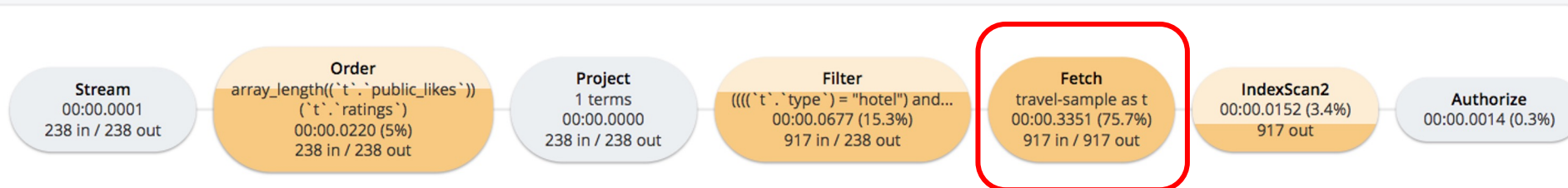
travel-sample.def_type

Buckets

travel-sample t

Fields

*travel-sample.type travel-sample.country travel-sample.public_likes
travel-sample.* t.public_likes t.ratings*



N1QL – Simple SELECT



- Run a basic query

```
SELECT * FROM `travel-sample` WHERE name = 'Atifly'
```

- Find out what is the key of the returned object (use META().id)
- Write a query that returns the same object using USE KEYS
- Observe the difference in query execution time

```
[
  {
    "travel-sample": {
      "callsign": "atifly",
      "country": "United States",
      "iata": "A1",
      "icao": "A1F",
      "id": 10226,
      "name": "Atifly",
      "type": "airline"
    }
  }
]
```

Impact of Indexes for Query Execution



- Try running the original query again:

```
SELECT * FROM `travel-sample` WHERE name = 'Atifly';
```

```
EXPLAIN SELECT * FROM `travel-sample` WHERE name = 'Atifly';
```

- Now, lets create an index on the name field:

```
CREATE INDEX idx_airline_name ON `travel-sample` (name);
```

- Now rerun the query:

```
SELECT * FROM `travel-sample` WHERE name = 'Atifly';
```

```
EXPLAIN SELECT * FROM `travel-sample` WHERE name = 'Atifly';
```

- Did the indexed field reduced the query time by an order of magnitude?

N1QL – JOIN



- Run a query with a JOIN

```
SELECT air.name, route.sourceairport,  
       route.destinationairport  
FROM `travel-sample` AS route  
JOIN `travel-sample` AS air  
ON KEYS route.airlineid  
LIMIT 2;
```

- Optional: Try out changing to **LEFT JOIN**.

Why may you get empty objects? Now try adding filter on **WHERE route.type = "route"**

```
[  
  {  
    "destinationairport": "ATL",  
    "name": "Air France",  
    "sourceairport": "XNA"  
  },  
  {  
    "destinationairport": "CDG",  
    "name": "Air France",  
    "sourceairport": "VIE"  
  }  
]
```


N1QL - Children in the Tree



- Run a basic query with nested Arrays

```
SELECT *  
FROM `travel-sample` AS route  
WHERE ANY child IN route.schedule SATISFIES  
child.flight = 'AF443' END LIMIT 2;
```

- Try to create the following array index and check how it improves query performance

```
CREATE INDEX idx_route_flights ON `travel-  
sample` (DISTINCT ARRAY child.flight FOR  
child IN schedule END);
```

- Optional: Modify the query to select routes that contain flights after 23:00:00 and create a suitable index

```
[  
  {  
    "route": {  
      "airline": "AF",  
      "airlineid": "airline_137",  
      "destinationairport": "MRS",  
      "distance": 2881.617376098415,  
      "equipment": "320",  
      "id": 10000,  
      "schedule": [  
        {  
          "day": 0,  
          "flight": "AF443",  
          "utc": "20:59:00"  
        },  
        ....  
      ]  
    }  
  }  
]
```

N1QL Queries over the Terminal



- SSH into server running the query service (using PuTTY or Mac Terminal):
 - `ssh root@192.168.61.101`
 - `couchbase123!`
- Start the Couchbase query shell and try to run a query
 - `/opt/couchbase/bin/cbq --user Administrator --password password`
 - `SELECT * FROM `travel-sample` WHERE name = 'Atifly';`

N1QL Query Service Over REST



- The Query Service (port 8093) responds to GET requests
- Try this from a terminal:
 - `curl -v http://192.168.61.101:8093/query/service -d 'statement=SELECT name FROM system:keyspaces&creds=[{"user": "Administrator", "pass": "password"}]'`
- The Query service always returns timing statistics for how long it took to retrieve the information, and to send it to you.
- For a more detailed look at what the query service is doing try passing the same query, with “explain”:
 - `curl -v http://192.168.61.101:8093/query/service -d 'statement=EXPLAIN SELECT name FROM system:keyspaces&creds=[{"user": "Administrator", "pass": "password"}]'`

Index Optimization

Lab Objectives



- Exploring different types of indexes – composite, partial and covering – and their effect on performance
- Observe pagination pushdown into index



Initial Query Performance



Open the query workbench and execute the following statement:

```
SELECT * FROM `travel-sample` t
WHERE type = "hotel" AND country = "United Kingdom" AND ARRAY_LENGTH(public_likes) > 3
ORDER BY ARRAY_LENGTH(public_likes), ratings DESC
OFFSET 0
LIMIT 20;
```

What is the execution time? What indexes are used?

Execute Explain success | elapsed: 325.29ms | execution: 325.27ms | count: 20 | size: 181449 [Preference](#)

Query Results JSON Table Tree Plan Plan Text

Indexes	Buckets	Fields
travel-sample.def_type	travel-sample t	travel-sample.type travel-sample.country travel-sample.public_likes travel-sample.* t.public_likes t.ratings

Stream 00:00.0000 20 in / 20 out

Limit 20 00:00.0000 20 in / 20 out

Order array_length(('t'.public_likes)) ('t'.ratings) 20 00:00.0083 (2.1%) 238 in / 20 out

Project 1 terms 00:00.0000 238 in / 238 out

Filter (((('t'.type') = "hotel") and... 00:00.0970 (24.9%) 917 in / 238 out

Fetch travel-sample as t 00:00.2727 (69.9%) 917 in / 917 out

IndexScan2 00:00.0071 (1.8%) 917 out

Authorize 00:00.0047 (1.2%)

The index returns 917 results before the filtering

Using a More Suitable Composite Index



Open the query workbench and execute the following statements (one at a time)

```
CREATE INDEX idx_hotel_ctry_likes
ON `travel-sample`(country, ARRAY_LENGTH(public_likes))
WHERE type = "hotel";

SELECT * FROM `travel-sample` t WHERE type = "hotel" AND country = "United Kingdom" AND
ARRAY_LENGTH(public_likes) > 3
ORDER BY ARRAY_LENGTH(public_likes), ratings DESC
OFFSET 0
LIMIT 20;
```

Why is the execution time better? Can it be improved?

Execute Explain success | elapsed: 106.96ms | execution: 106.95ms | count: 20 | size: 181449

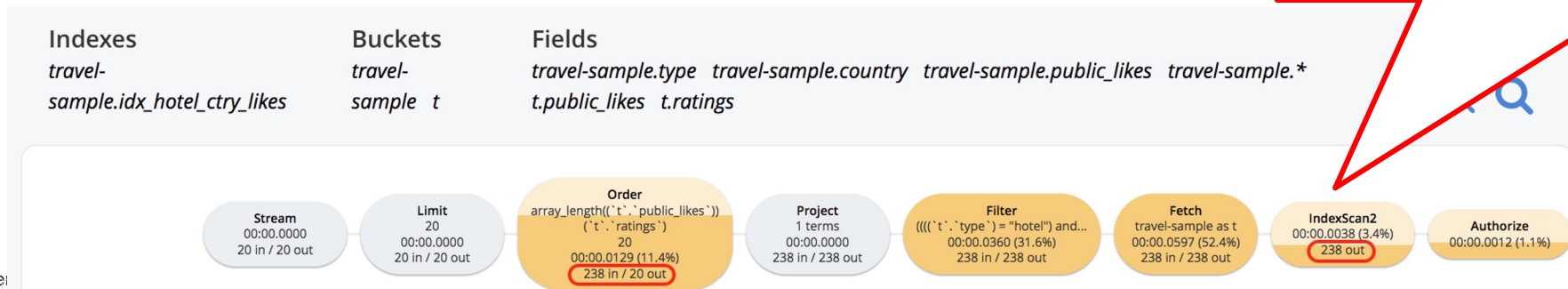
Query Results

JSON

Table

T

The index returns 238 results before the ordering because the DESC order is not pushed down the index



Using a Partial Index



Open the query workbench and execute the following statement (one at a time):

```
DROP INDEX `travel-sample`.idx_hotel_ctry_likes;

CREATE INDEX idx_hotel_ctry_likes_ratings
ON `travel-sample`(country, ARRAY_LENGTH(public_likes), ratings DESC)
WHERE type = "hotel";
```

Rerun the previous query again. Why is the query execution time better?

Execute

Explain

success | elapsed: 20.31ms | execution: 20.28ms | count: 20 | size: 181449

Preferences

Query Results

JSONTableTreePlanPlan Text

Indexes	Buckets	Fields
travel-sample.idx_hotel_ctry_likes_ratings	travel-sample	travel-sample.type travel-sample.country travel-sample.public_likes travel-sample.*

Stream

Limit

Project

Filter

Fetch

IndexScan2

```
"#operator": "IndexScan2",
"#stats": {
  "index": "idx_hotel_ctry_likes_ratings",
  "index_id": "1e1bb81a3f2e21c8",
  "index_projection": {
    "primary_key": true
  },
  "keyspace": "travel-sample",
  "limit": "20",
```

Limit has been pushed down the index.

Offset & Limit Push Down to Index Scan



Rerun the previous query with different offsets (40, 100, 200) and compare the results.

```
1 SELECT * FROM `travel-sample` t WHERE type = "hotel" AND country = "United Kingdom"
2 AND ARRAY_LENGTH(public_likes) > 3
3 ORDER BY ARRAY_LENGTH(public_likes), ratings DESC
4 OFFSET 100
5 LIMIT 20;
```

Execute Explain success | elapsed: 23.57ms | execution: 23.55ms | count: 20 | size: 178376 Preferences

Query Results JSON Table Tree Plan Plan Text

Indexes	Buckets	Fields
travel-sample.idx_hotel_ctry_likes_ratings	travel-sample	travel-sample.type travel-sample.country travel-sample.public_likes travel-sample.*

Limit 20 00:00.0000 (0.2%) 20 in / 20 out

Project 1 terms 00:00.0000 20 in / 20 out

Filter (((('t'.type) = "hotel") and... 00:00.0031 (14.1%) 20 in / 20 out

Fetch travel-sample as t 00:00.0155 (69.6%) 20 in / 20 out

IndexScan2 20 00:00.0020 (9%) 20 out

Authorize 00:00.0015 (6.9%)

Execution time is not related to offset. Execution time are the same (offset=0 or offset=100) and independent from the offset value because the offset has been pushed down.

Using a Covering Index



Execute the following statements (one at a time)

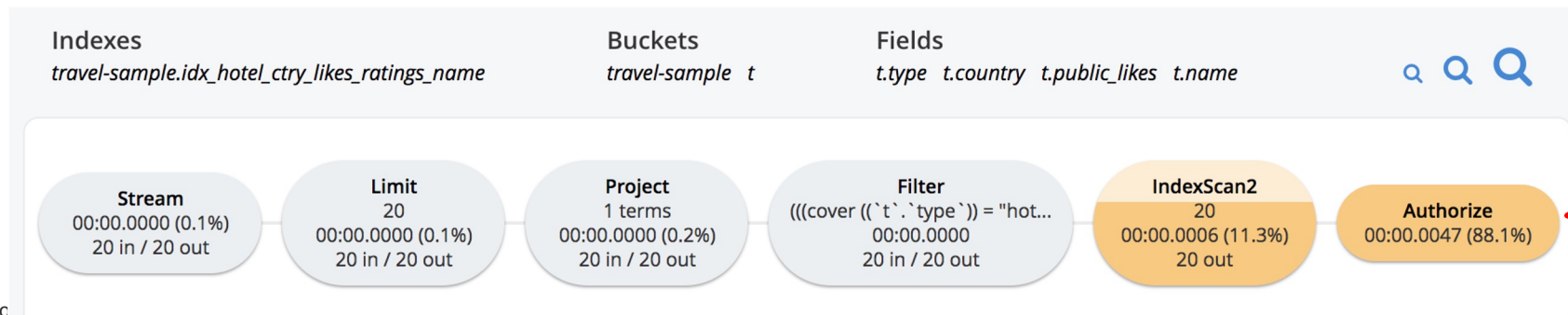
```
DROP INDEX `travel-sample`.idx_hotel_ctype_likes_ratings;  
CREATE INDEX `idx_hotel_ctype_likes_ratings_name`  
ON `travel-sample`(`country`,array_length(`public_likes`),`ratings` DESC, name)  
WHERE (`type` = "hotel");  
SELECT name FROM `travel-sample` t WHERE type = "hotel" AND country = "United Kingdom"  
AND ARRAY_LENGTH(public_likes) > 3  
ORDER BY ARRAY_LENGTH(public_likes), ratings DESC  
OFFSET 0 LIMIT 20;
```

Why is the query so much faster now?

Execute Explain success | elapsed: 6.95ms | execution: 6.93ms | count: 20 | size: 1056 Preferences

Query Results

JSON Table Tree Plan Plan Text



FTS Basics

FTS: Create a default Index



Let's build the default Index on `travel-sample` bucket.

- Build a new default Index
 - Give index name "idx_default"
 - Select bucket "travel-sample"
 - Create Index
- Observe time to create default index. (~60s)
- Check the size on disk of the default index. (~ 915Mb)
- Search default index & review search results

Name idx_default **Bucket** travel-sample

Type Identifier

- ☒ JSON type field: type
- ☐ Doc ID up to separator: delimiter
- ☐ Doc ID with regex: regular expression

Type Mappings

- ☒ # default | dynamic

Analyzers

Custom Filters

Advanced

Index Replicas 0

Create Index Cancel

Full Text Indexes				Add Index
index name	bucket	doc count	indexing progress	
idx_default	travel-sample	31591	100%	
search this index...		Search	Delete	Clone Edit

Note: Default index are not recommended in a production environment

FTS: Create a custom Index on type = hotel



Let's build a new Index on `travel-sample` bucket.

- Build a new Index
 - Give index name "idx_hotel"
 - Select bucket "travel-sample"
 - Add a Type mapping = hotel (only index specified fields)
 - Disable the default mapping.
 - Add a child mapping on reviews (array)
 - Add a field mapping on content (only index specified fields and store)
- Observe time to create default index. (~5s)
- Check the size on disk of the default index. (~ 79Mb)
- Search "friendly" in the UI with the hotel index & review search results

Name
idx_hotel

Bucket
travel-sample

Type Identifier
☒ JSON type field: type
☐ Doc ID up to separator: delimiter
☐ Doc ID with regex: regular expression

▼ **Type Mappings**

<input checked="" type="checkbox"/>	# hotel	only index specified fields
<input checked="" type="checkbox"/>	{ reviews	dynamic
	content	text index store include in_all field include term vectors
<input type="checkbox"/>	# default	disabled dynamic

Bonus: Run the same search with the REST API. (you can find the curl command in the UI)