



Couchbase

Architecture and Administration Basics

N1QL & FTS Labs



1

Lab 1-N1QL Basics

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 ▾

IMPORT EXPORT

CouchbaseDay > Query

Query History

filter queries X

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

Plan Plan Text

Bucket Insights ↗

Fully Queryable Buckets

▶ travel-sample (31591)

Queryable on Indexed Fields

Non-Indexed Buckets

Dashboard

Servers

Buckets

Indexes

Search

Query

Execute Explain

XDCR

Security

Settings

Logs

Query Results

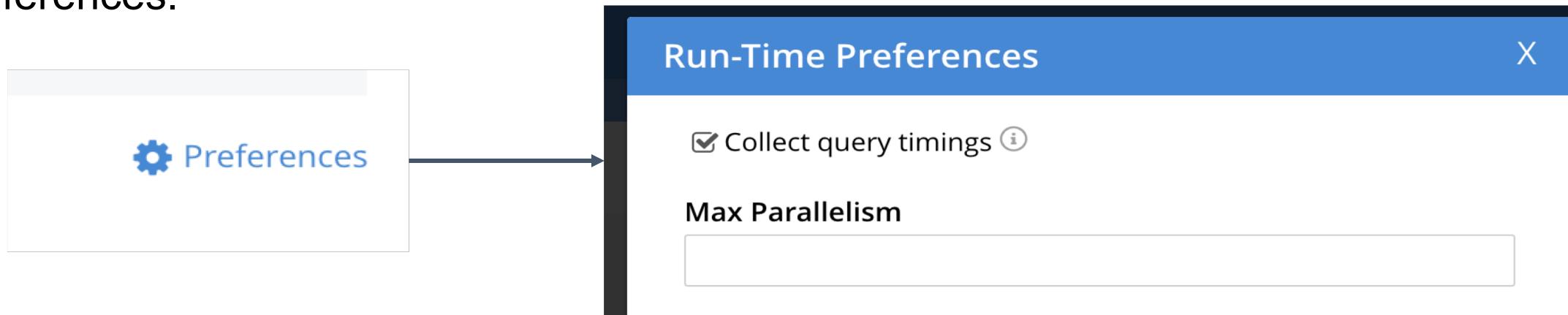
1 {"no_data_yet": "Hit execute to run query"}

Delete Selected Delete All

The screenshot shows the Couchbase Query Workbench interface. The left sidebar has a red box around the 'Query' tab. The main area shows a 'Query History' panel with five entries. The first four entries are 'Create index' commands for 'idx_city_replicated' on the 'travel-sample' bucket, with replica counts of 2, 1, 2, and 2 respectively. The fifth entry is a placeholder '5'. Below the history is a 'Plan' and 'Plan Text' section. To the right is a 'Bucket Insights' panel with sections for 'Fully Queryable Buckets' (listing 'travel-sample'), 'Queryable on Indexed Fields', and 'Non-Indexed Buckets'. The top navigation bar includes 'BETA BUILD', 'Activity', 'Documentation', 'Support', 'Administrator', 'IMPORT', and 'EXPORT'.

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 ?



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"}]'`



2

Lab 2: 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*

```
graph LR
    Stream((Stream  
0.00000  
0 / 20 out)) --> Limit((Limit  
20  
00:00.0000  
20 in / 20 out))
    Limit --> Order((Order  
array_length(`t`.`public_likes`))  
(`t`.`ratings`)  
20  
00:00.083 (2.1%)  
238 in / 20 out))
    Order --> Project((Project  
1 terms  
00:00.0000  
238 in / 238 out))
    Project --> Filter((Filter  
(((`t`.`type` = "hotel") and...  
00:00.0970 (24.9%)  
917 in / 238 out))
    Filter --> Fetch((Fetch  
travel-sample as t  
00:00.2727 (69.9%)  
917 in / 917 out))
    Fetch --> IndexScan2((IndexScan2  
00:00.0071 (1.8%)  
917 out))
    IndexScan2 --> Authorize((Authorize  
00:00.0047 (1.2%)))
```

The index returns 917 results before the filtering

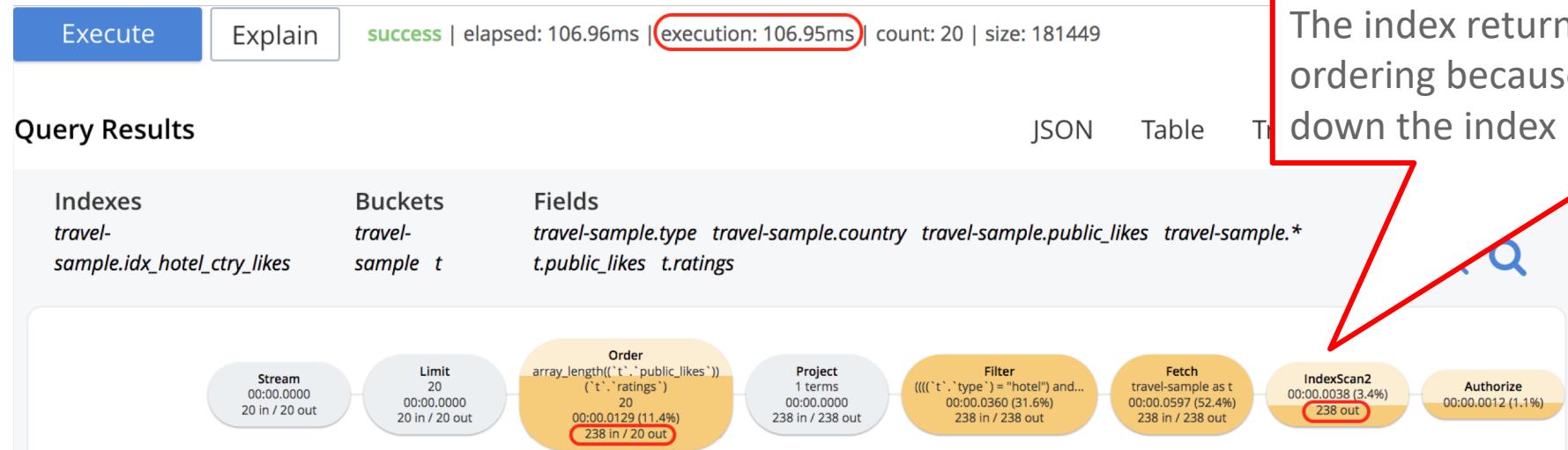
Using a More Suitable Composite Index

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?



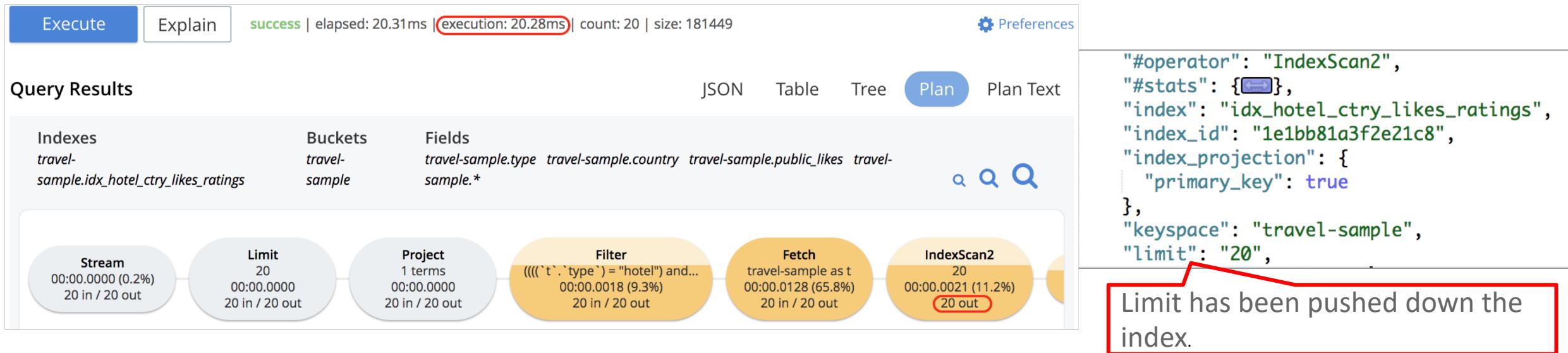
Using a Partial Index

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?



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

```
graph LR; Plan --> Limit[Limit 20]; Limit --> Project[Project 1 terms]; Project --> Filter[Filter (((`t`.`type` = "hotel") and...); Filter --> Fetch[Fetch travel-sample as t]; Fetch --> IndexScan2[IndexScan2 20]; IndexScan2 --> Authorize[Authorize 00:00.0015 (6.9%)];
```

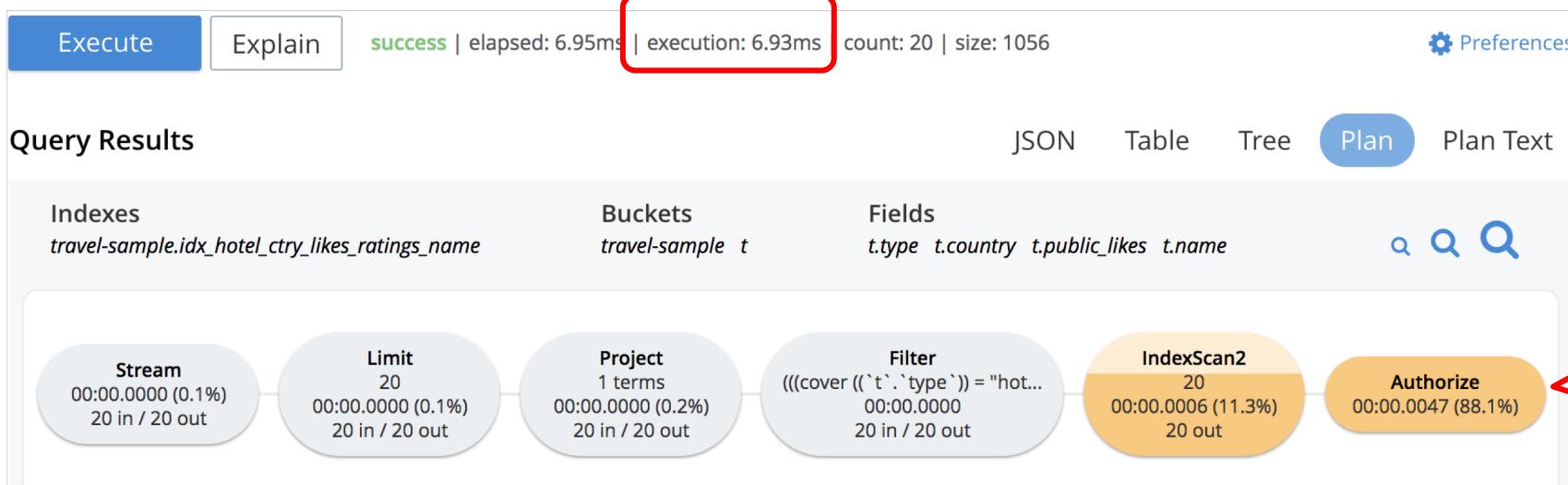
Execution time is not related to offset. Execution time are the same (offset=0 or offset=100) and independent from the offset value because the offset has been pushed down.

Using a Covering Index

Execute the following statements (one at a time)

```
DROP INDEX `travel-sample`.idx_hotel_ctry_likes_ratings;  
  
CREATE INDEX `idx_hotel_ctry_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?





3

Lab 3: 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

The screenshot shows the 'Create Index' dialog box. At the top, there are fields for 'Name' (set to 'idx_default') and 'Bucket' (set to 'travel-sample'). A red box highlights these two fields. Below this, there is a 'Type Identifier' section with three radio button options: 'JSON type field:' (selected), 'Doc ID up to separator:', and 'Doc ID with regex:'. Under 'Type Mappings', a single mapping is listed: '# default | dynamic' (with a checked checkbox). There are also sections for 'Analyzers', 'Custom Filters', and 'Advanced'. In the 'Index Replicas' section, a dropdown menu is set to '0'. At the bottom right, there are 'Create Index' and 'Cancel' buttons, with the 'Create Index' button being highlighted by a red box.

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	Bucket
idx_hotel	travel-sample
Type Identifier	
<input checked="" type="radio"/> JSON type field:	type
<input type="radio"/> Doc ID up to separator:	delimiter
<input type="radio"/> 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)