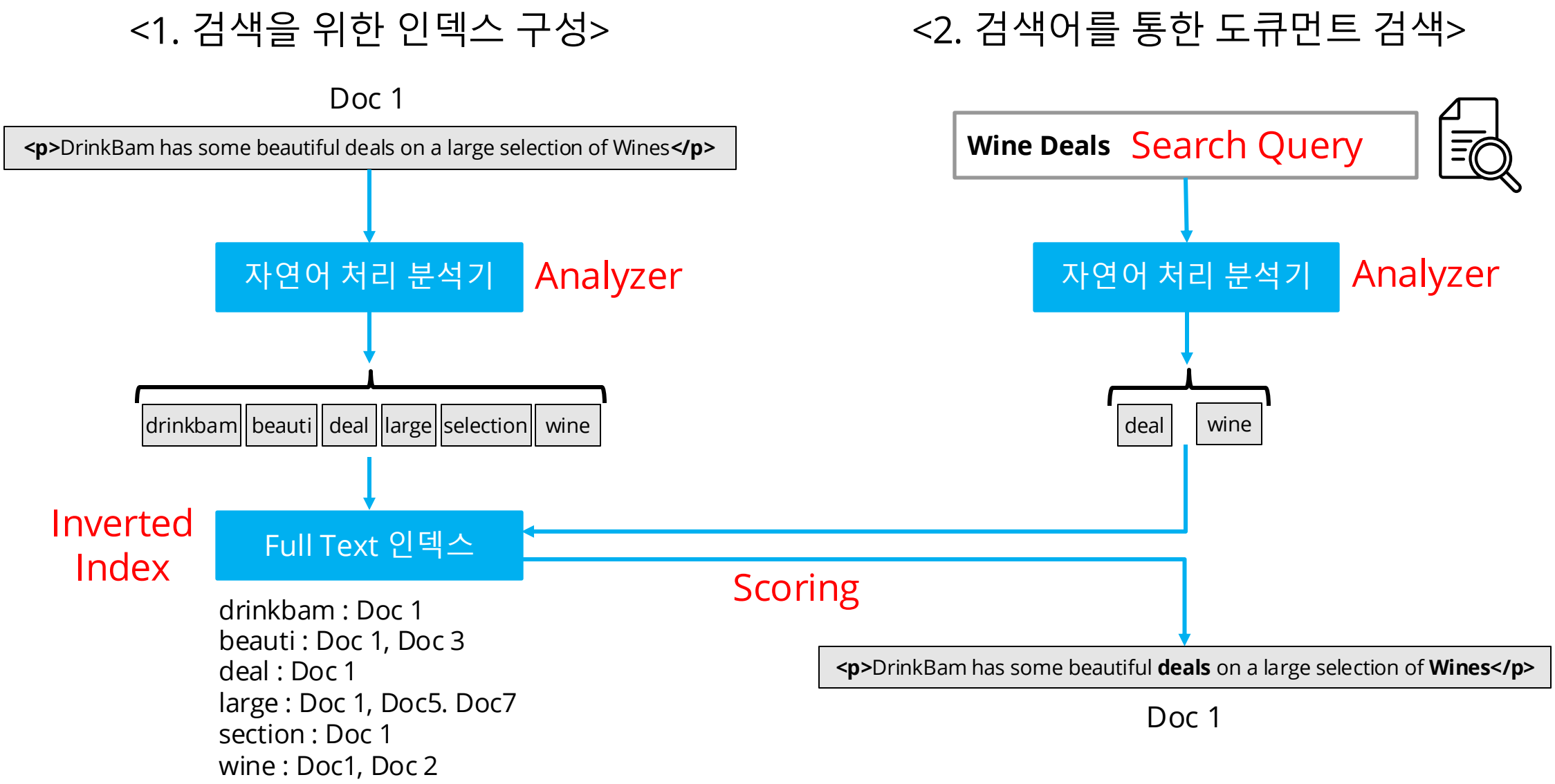
Couchbase

4교시.
Full Text Search 개요, 실습

1 FTS 개요
2 Search Query
3 Search Results
4 Search Indexes
5 FTS 실습

4-1.Full Text Search 개요

Couchbase의 텍스트 검색 엔진



Analyzer : 검색에 맞는 언어 식별성

Raw 텍스트 데이터

DrinkBam has some beautiful deals on a large selection of Wines

Character filters
불필요한 글자(구두점, 공백) 및 Tag 등을 제거하기 위한 필터

DrinkBam has some beautiful deals on a large selection of Wines

Tokenizers
전체 텍스트를 개별 Token 형태로 분리

DrinkBam | has | some | beautiful | deals | on | a | large | selection | of | Wines

Token Filters
분리된 Token에서 stop word(a, the 등) 제거, 기본형 변환

drinkbam | has | some | beautiful | deals | on | a | large | selection | of | wines

drinkbam | beautiful | deals | large | selection | wines

최종 분석완료 Token

drinkbam | beauti | deal | large | selection | wine

Inverted Index : 검색 성능

<Raw 텍스트 데이터>

doc1	**<p**>The quick brown fox</**p>**
doc2	**<p**>The quick brown fox jumps over the lazy dog</**p>**
doc3	**<p**>The quick brown fox jumps over the quick dog</**p>**
doc4	**<p**>Brown fox brown dog</**p>**
doc5	**<p**>Lazy jumping dog</**p**>

Analyzer

<Inverted Index>

Terms	도큐먼트ID
brown	doc1, doc2, doc3, doc4
dog	doc2, doc3, doc4, doc5
fox	doc1, doc2, doc3, doc4
jump	doc2, doc3, doc5
lazi	doc2, doc5
over	doc2, doc3
quick	doc1, doc2, doc3

Search Query : 검색 용이성

quick dog

SEARCH

{"field": "Search_Terms",
"**match**": "quick dog"}

Analyzer

quick | dog

<Inverted Index>

Terms	도큐먼트ID
brown	doc1, doc2, doc3, doc4
dog	doc2, doc3, doc4, doc5
fox	doc1, doc2, doc3, doc4
jump	doc2, doc3, doc5
lazi	doc2, doc5
over	doc2, doc3
quick	doc1, doc2, doc3

<Search Result>

doc3	** **The **quick** brown fox jumps over the **quick dog **
doc2	** **The **quick** brown fox jumps over the lazy **dog **
doc1	** **The quick brown fox** **
doc5	** **Lazy jumping dog** **
doc4	** **Brown fox brown dog** **

Scoring : 검색 정확도

Fox are jumping

SEARCH

{"field": “Search_Terms",
"**match**": “Fox are jumping"}

- **TF(Term Frequency) : 개별 도큐먼트 내에 해당 Token이 자주 나올수록 높은 점수**
- **IDF(Inverse Document Frequency) : 해당 Token이 포함된 도큐먼트 개수가 많을수록 낮은 점수**

<Search Result>

doc3	<**p**>The **quick** brown fox jumps over the **quick dog**</**p**>	**Score : 0.8762741**
doc2	<**p**>The **quick** brown fox jumps over the lazy **dog**</**p**>	**Score : 0.6744513**
doc1	<**p**>The quick brown fox</**p**>	**Score : 0.6173784**
doc5	<**p**>Lazy jumping dog</**p**>	**Score : 0.35847884**
doc4	<**p**>Brown fox brown dog</**p**>	**Score : 0.32951736**

Score = TF / IDF

4-2. Search Queries

Query types

- Simple Queries
- Compound Queries
- Range Queries (string, date, numeric)
- String Queries (natural language)
- Geospatial Queries
- Non-analytic (i.e. exact match)
- Special queries (for dev purpose)
- Vector queries

Interacting with FTS

RESTful API

```
{
  "match": "location hostel",
  "field": "reviews.content",
  "analyzer": "standard",
  "fuzziness": 2,
  "prefix_length": 4,
  "operator": "and"
}
```

SDK Clients

```
public static void simpleTextQuery(Bucket bucket) {
        String indexName = "travel-sample-index";
        MatchQuery query = SearchQuery.match("swanky");
        SearchQueryResult result = bucket.query(new
                SearchQuery(indexName, query).limit(10));
        printResult("Simple Text Query", result);
}
```

Through N1QL
Search or FLEX Indexing

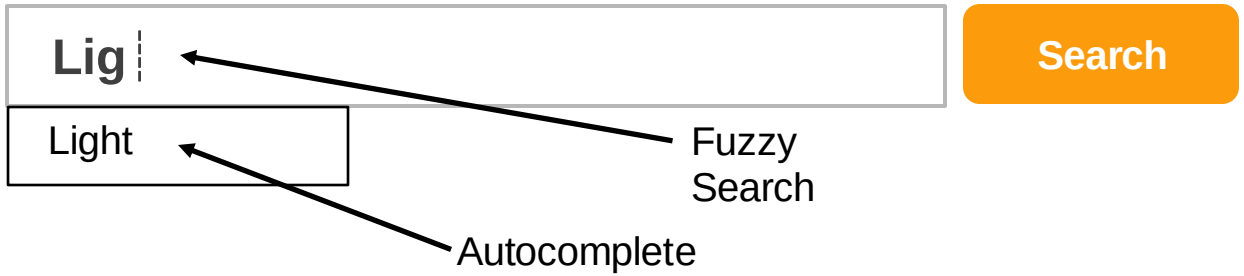
```
SELECT t1.name
FROM `travel-sample` AS t1
WHERE SEARCH(t1, {
     "match": "bathrobes",
     "field": "reviews.content",
     "analyzer": "standard"
});
```

```
SELECT META(b).id
FROM mybucket AS b
USE INDEX (USING FTS)
WHERE b.f1 = "xyz"
     AND b.f2 = 100;
```

Simple Queries

Query type	Example	Matched terms results
MATCH Result fields	{"field": "reviews.content","**match**": "beautiful"} { "query": {"match": "beautiful","field": "reviews.content"}, "fields": ["reviews.content", "name"] }	Beauty, beautiful
MATCH PHRASE	{"field": "reviews.content","**match_phrase**": "beautiful location"}	beautifully located beautiful location
FUZZY Result fields Highlight	{"field": "reviews.content", "**term**": "hotel", "**fuzziness**":1} { "query": {"field": "reviews.content", "**term**": "hotel", "**fuzziness**":1}, "fields": ["reviews.content", "name"] } { "query": {"field": "reviews.content", "**term**": "hotel", "**fuzziness**":1}, "fields": ["name", "reviews.content"], "highlight": {} }	hotel hostel
PREFIX	{"field": "reviews.content", "**prefix**":"bea"}	Beach, beautiful
REGEXP	{"field": "reviews.content","**regexp**":"ho[st|t]el"}	Hostel, hotel
WILDCARD Result fields Highlight	{"field": "reviews.content","**wildcard**":"ho?tel"} { "query": {"field": "reviews.content","**wildcard**":"ho?tel"}, "fields": ["reviews.content", "name"]} { "query": {"field": "reviews.content","**wildcard**":"ho?tel"}, "fields": ["reviews.content", "name"], "highlight": {} }	hostel
BOOLEAN FIELD	{"field": "reviews.content","**bool**": true,"**field**":"free_breakfast"}	*documents where the field contains boolean true value*

Fuzziness



Fuzziness allows you to find words when the search term is misspelled.

- The parameter indicates how many letters may be different or missing, leveraging the Levenshtein distance.
- Maximum supported fuzziness is 2 to lower the number of false positives
- One important optimization is that most spelling mistakes happen towards the end => utilize the *prefix_length* option in fuzzy queries

```
{
  "Match":"beautiful",
  "field":"reviews.content",
  "analyzer":"standard",
  "fuzziness":1,
  "prefix_length":2
}
```

"autiful" is only considered for fuzziness

Compound Queries

Query type	Description	Example
CONJUNCTION	Logical AND. Contains multiple child queries. Its result documents must satisfy all of the child queries.	{ "**conjuncts**": [{"field": "reviews.content", "match": "location"}, {"field": "free_breakfast", "bool": true}]}
DISJUNCTION	Logical OR. Contains multiple child queries. Its result documents must satisfy a configurable `min` number of child queries. By default this `min` is set to 1.	{"**disjuncts**": [{"field": "reviews.content", "match": "location"}, {"field": "free_breakfast", "bool": true}]}

BOOLEAN	Combination of conjunction and disjunction queries and takes three lists of queries:
	• `must`: Result documents must satisfy all of these queries.
	• `should`: Result documents should satisfy these queries.
	• `must not`: Result documents must not satisfy any of these queries.
		{"**must**": {
		"conjuncts": [
		{"field": "reviews.content",
		"match": "location"}]},
		"must_not": {
		"disjuncts": [
		{"field": "free_breakfast",
		"bool": false}]},
		"should": {
		"disjuncts": [
		{"field": "free_breakfast",
		"bool": true}]}}

DOC ID	Returns the indexed document or documents among the specified set. This is typically used in conjunction queries, to restrict the scope of other queries' output.	{ "**ids**": ["hotel_10158", "hotel_10159"] }

Boosting

Boosting allows you to give different weights to each element in a compound query.

```
{
   "conjuncts": [
     {
         "field": "description",
               "match": "pool"
         },
     {
         "field": "reviews",
              "match":”pool”,
         “boost”:2
     }
  ]
}
```

Performs Match Queries for `pool` in both the `reviews` and `description` fields, but documents having the term in the `reviews` field score higher.

Range Queries

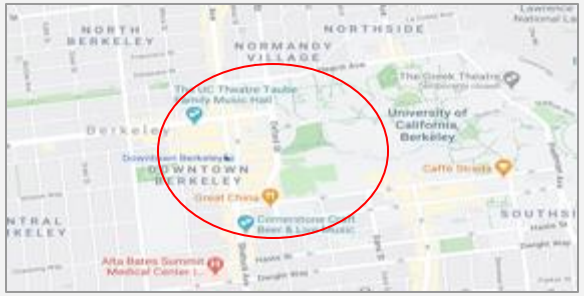
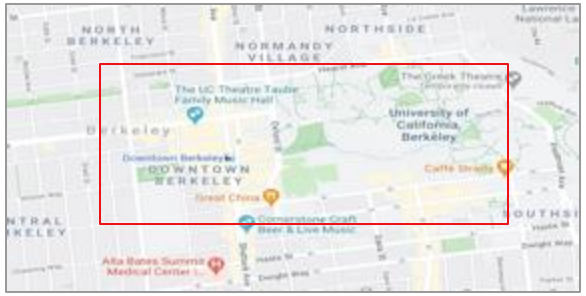
Query type	Description	Example
DATE RANGE	Finds documents containing a date value, in the specified field within the specified range.	{ "start": "2001-10-09T10:20:30-08:00", "end": "2016-10-31", "inclusive_start": false, "inclusive_end": false, "field": "review_date" }
NUMERIC RANGE	Finds documents containing a numeric value in the specified field within the specified range	{ "min": 100, "max": 1000, "inclusive_min": false, "inclusive_max": false, "field": "id" }
TERM RANGE	Finds documents containing a term in the specified field within the specified range.	{ "min": "foo", "max": "foof", "inclusive_min": false, "inclusive_max": false, "field": "desc" }

Query String

Query type	Description	Example
QUERY STRING	Query strings enable humans to describe complex queries using a simple syntax.	{ "**query**": "+nice +view" }

Example with Query String syntax	Same example with Query syntax
{ "**query**": “pool" }	{ "**match**": "pool", “field”:”_all”}
{ "**query**": "continental breakfast"}	{ "**match_phrase**": "continental breakfast" , “field”:”_all”}
{ "**query**": "description:pool" }	{ "**match**": "pool", “field”:”description”}
{ "**query**": "+description:pool - continental breakfast" }	{"**must**": { "conjuncts": [{"field":"description", "match":"pool"}]}, **"must_not"**: { "disjuncts": [{"field":"default","match":“continental”}]}, **"should"**: { "disjuncts": [{"field":"default", "match":“breakfast”}]}}
{ "**query**":"description:pool name:pool^5”}	*Boost documents having a match on name:pool*

GeoSpatial Query

Query type	Description	Example
POINT DISTANCE		{“field”:”geo”, “**location**”:”53.482358,-2.235143"”, “**kilometers**”,”1” }
BOUNDED RECTANGLE		{“field”:”geo”, "**top_left**": [-2.235143, 53.482358], "**bottom_right**": [28.955043, 40.991862], }
BOUNDED POLYGON		{"field": "geo", "**polygon_points**": ["37.79393211306212,-122.44234633404847", "37.77995881733997,-122.43977141339417", "37.78803109202015 5,-122.42925715405579", "37.79026946582319,-122.41149020154114", "37.79571192027403,-122.40735054016113"] }

N1QL with SEARCH predicate

Identify the customer accounts and their related contacts where a particular topic has been discussed. The search criteria may include one or many of the following informations: meeting Title, Date range, Customer Contact Details, Sales team member details

```
SELECT meta(a).id, a.title,a.startDate, a.account.name, a.contacts, a.participants
FROM crm a
WHERE SEARCH(a,
    {"conjuncts": [
        {"field":"title", "match": "artificial intelligence"}   ,
        {"field":"participants.name", "match":"james"}   ,
        {"field":"account.name", "match":"willis"}  ,
        {"field":"startDate","start": "2019-03-20", "end":"2019-03-31"} ,
        {"field":"contacts.name", "match":"boone"}   ,
        {"field":"contacts.email", "match":"obell@gmail.com"}
      ]
    },
    {"index":"all_acts"}
  )
  AND a.type='activity'
  AND a.activityType='Appointment'
```

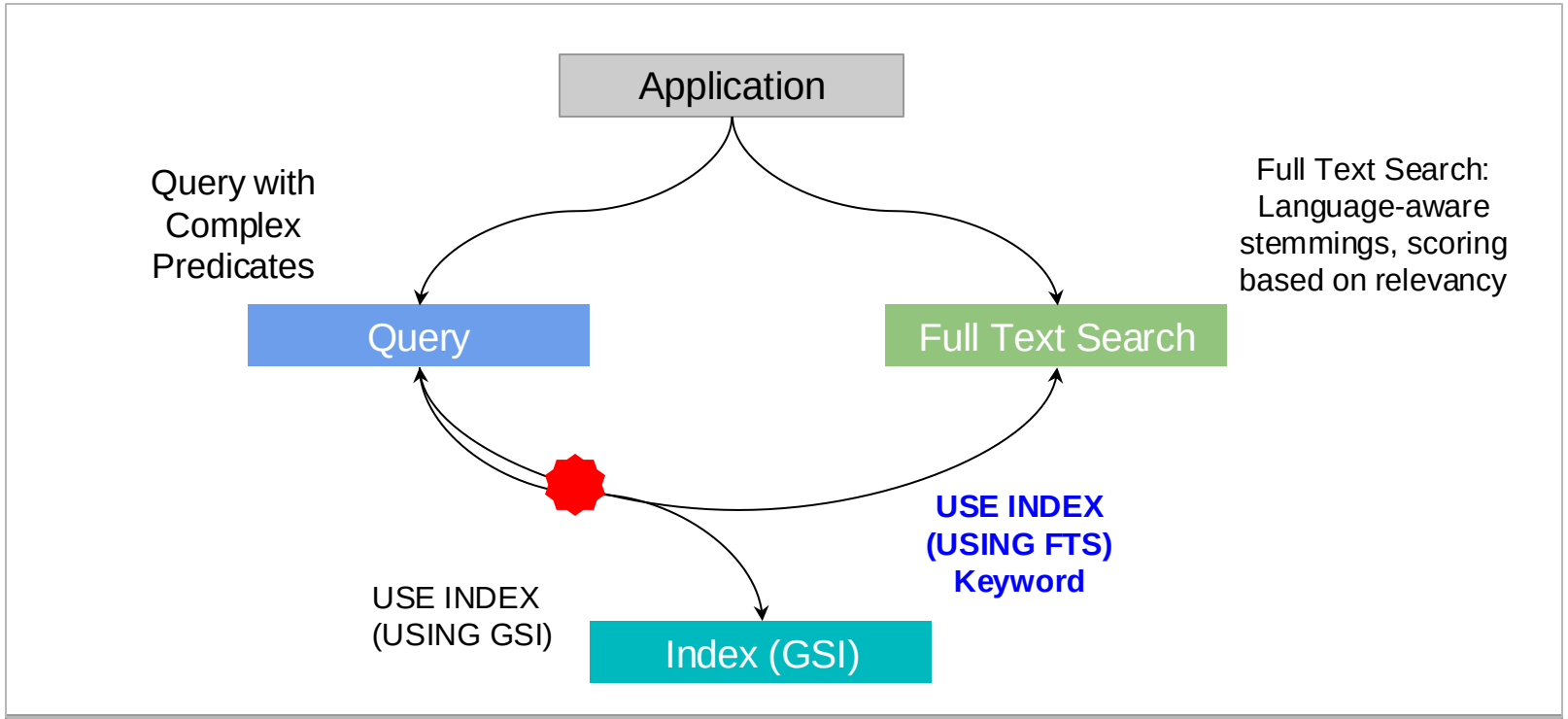
SQL clause

Search index tip

Search clause

N1QL with FLEX Indexing

- Mechanism whereby a N1QL query can leverage either or both Secondary Index and Keyword Text Search with standard N1QL predicates
- Allows N1QL to transparently benefit the full power Text Search capability without any Query limitations
  - > Nested document field, Array, SQL Aggregation, Join and Sorting



FLEX Index allows the Query to leverage either GSI or FTS

N1QL with FLEX Indexing When should you use them?

- Where the search conditions of the N1QL statements are **not predetermined**
  - They can contain varied numbers of predicates, often based on user's selections
  - It is difficult to create indexes to cover all of the search conditions.
- Applications that provide search capabilities involving a **large number of predicates**
  - With logical operators, such as AND/OR combinations in the search conditions.
- Where the search conditions involve predicates on **hierarchical document elements**
  - Such as search that involve array elements in an array, or in multiple arrays.
- Where the applications require **both the power of FTS and need SQL aggregation**, with JOIN to include related information from other objects.
- Or you simply want to use the **N1QL predicate syntax** over the FTS syntax

N1QL with FLEX Indexing Example

There are two ways to specify that you would like to use a full-text index with a N1QL query without search predicate:

- Use the `USE FTS` hint in the N1QL query.
- Set the `use_fts` request-level parameter to `true`.

```
SELECT META(b).id
FROM mybucket AS b
USE INDEX (USING FTS)
WHERE b.f1 = "xyz" AND b.f2 = 100;
```

1. The query engine considers all available full-text indexes.
2. If any full-text index qualifies, the full-text index is used.
3. If none of the full-text indexes qualify, the query engine considers other available GSI and primary indexes, following existing rules.

Requirements	Description
FTS index	Analysers, Type mappings and Indexed fields must satisfy requirements listed in the documentation.
Query	In order to use a full-text index with a N1QL query, the query predicates (conditions and expressions) must also meet certain requirements listed in the documentation.

4-3. Search Results

Sorting

- The required sort-type is specified by using the `sort` field, as an array of String or Objects. Combination is possible.
- The default sort-order is ascending.
- If multiple fields are included, the sorting of documents begins according to their values for the field whose name is first in the array.

Sorting type	Description	Example

Sorting with Strings	Array of strings containing either:
	• field name
	• _id:
	• _score
		{
		"fields": ["title"],
		"**sort**": ["country", "-_score","-_id"],
		"query":{"query": "beautiful pool"}
		}

Sorting with objects	Fine-grained control over sort-procedure. Each object can have the following fields:
	• by
	• field
	• missing
	• Mode
	• type
		{
		...
		"**sort**": [
		"country",
		{
		"by" : "field",
		"field" : "reviews.ratings.Overall",
		"mode" : "max",
		"missing" : "last",
		"type": "number"
		}
		}

Pagination

Pagination of large number of results are essential for sorting and displaying a subset of these results.

Pagination type	Description	Example
SIZE/LIMIT, FROM/OFFSET	To obtain a subset of results and works deterministically when combined with a certain sort order (default order is relevance)	{ "query": { "match": "California", "field": "state" }, "size": 5, "**from**": 10 }
SEARCH_AFTER SEARCH_BEFORE	For more efficient pagination, designed to fetch the size number of results after or before the key specified. Allow for the client to maintain state while paginating.	{ "query": { "match": "California", "field": "state" }, "sort": ["_id"], "**search_after**": ["hotel_10180"], "size": 3 }

Facets

Facets are **aggregate** information collected on a particular search result set. You do a search and collect additional facet information along with it.

Facet type	Description
TERM FACET	Counts up how many of the matching documents have a particular term in a particular field
NUMERIC RANGE FACET	The user defining their numeric ranges. The facet counts matching documents for a particular field.
DATA RANGE FACET	Same as numeric range facet, but on dates instead of numbers

Term Facet example : search documents containing `air fresheners` and compute facets on the field `type`

```
{
    "query": {"query": "air fresheners"},
    "facets": {
        "type": {"field": "type"}
    }
}
```

Results : documents containing `air fresheners` can be grouped in 2 facets: `Electric Air Fresheners` (288 results) and `Air Fresheners Sprays` (887 results)

```
"facets": {
    "type": {
        "field": "type",
        "total": 1175,
        "terms": [
            {"term": "Electric Air Fresheners","count": 288},
            {"term": "Air Freshener Sprays", "count": 887}
        ]
    }
}
```

4-4. Search Indexes

Index Overview

Real-time indexing (auto-updated upon mutation)

Mapping (default map, map by document type and dynamic mapping)

Storing (Stored fields, Term vectors)

Analyzers (Character filters, Tokenization, Token Filtering)

Aliasing (Searches can be performed across multiple buckets)

Creating Search Index

Couchbase
Web Console

RESTful
API

```
{
  "name":"demoIndex",
  "type":"fulltext-index",
  “params":{ ...},,
  "sourceType":"couchbase",
  "sourceName":"travel-sample",
  "sourceUUID":"99e9829898a45ba35f1c9c85dfcdb42b",
  "sourceParams":{...},
  "planParams":{},
  "uuid":""
}
```

TIP: The easiest way to create a FTS index is through the Web Console, then the Index Definition Preview can be copy/paste and reused with REST API

Creating index

Define your indexes based on your access pattern.

1

Identifying document type format

Type Identifier (Json property, Doc ID)

2

Include or Exclude documents by type

Type Mapping (all fields or only specific fields)

Analyser (default, specific or custom)

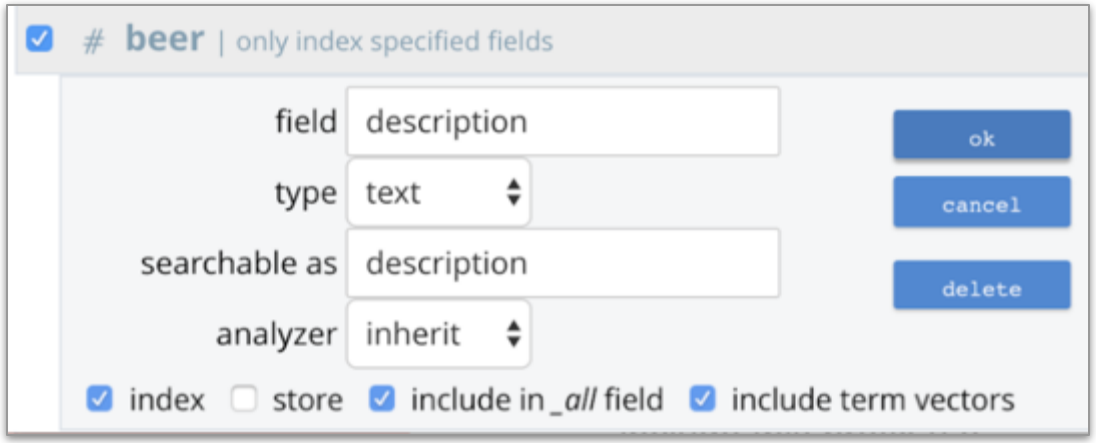
3

Include or Exclude specific fields in the index

Child Field (value or array) or Child Mapping (Json object)

Store (for highlights in results)

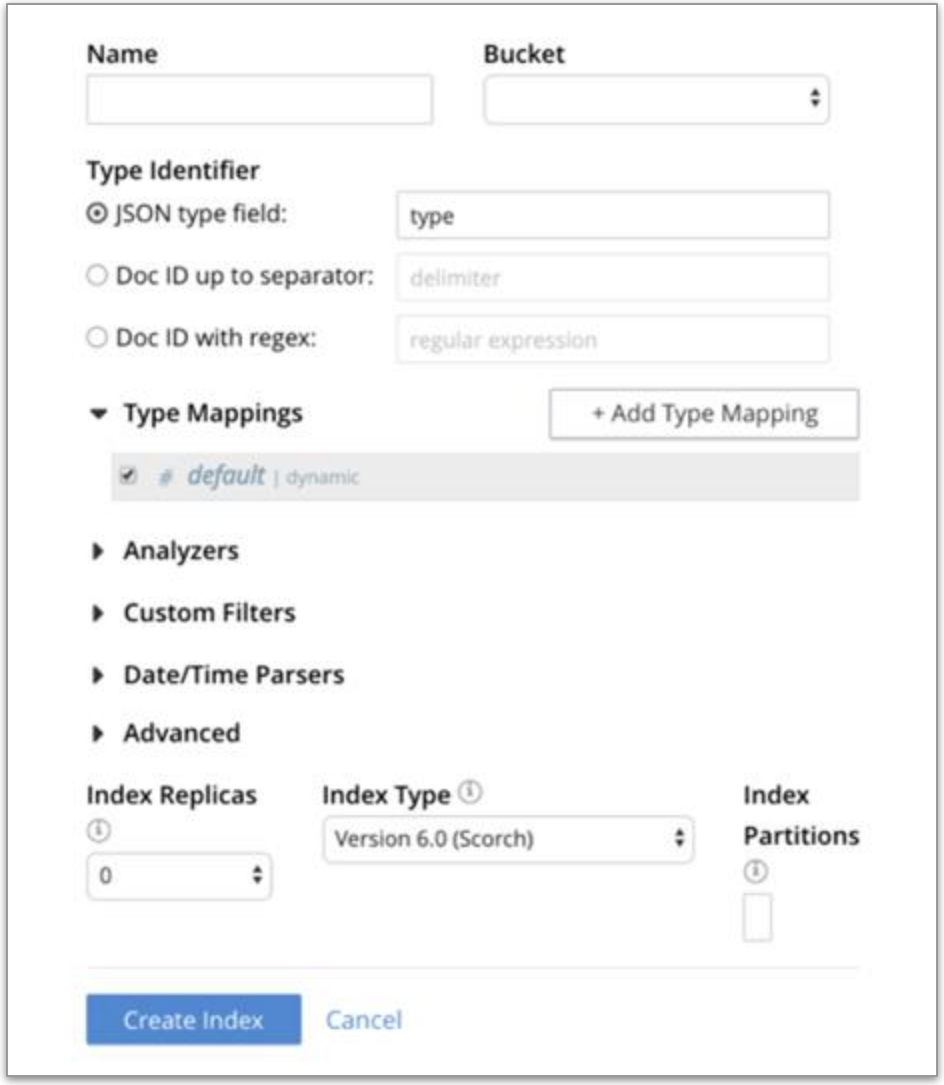
Index mapping



OPTION name	Description
index	If unchecked, fields that match this will not be indexed
store	• This allows the document contents to be written to the index; by default only doc IDs are written to a FTS Index • Enables highlighting and result snippets but generally results in larger indexes that are slower to build. • Encourage use of multi-gets so users don't need to store the additional information in the index.
Include in _all	• The text in this field will be searchable in query strings without prefixing the field name. • If unchecked, the query must include this prefix (i.e. "desc:modern")
Include term vectors	Not storing term vectors results in smaller indexes and faster index build times.

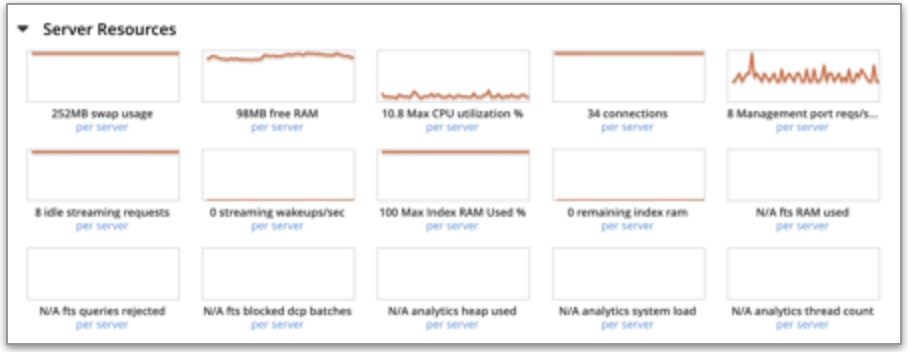
Search Workbench - Creating Indexes

- Visual Web Console
- All features available, including Custom Analyzers creation
- Index definition available in JSON for REST API
- Once built, searchable from Web Console, SDK or REST API



Monitoring Search

Couchbase
Web Console



RESTful
API


```
{
  "num_bytes_used_ram": 213924088,
  "travel-sample:geoIdx:avg_queries_latency": 41.771365,
  "travel-sample:geoIdx:num_bytes_used_disk": 295152367,
  "travel-sample:geoIdx:total_queries": 9,
  "travel-sample:geoIdx:total_queries_slow": 0,
  ...
}
```

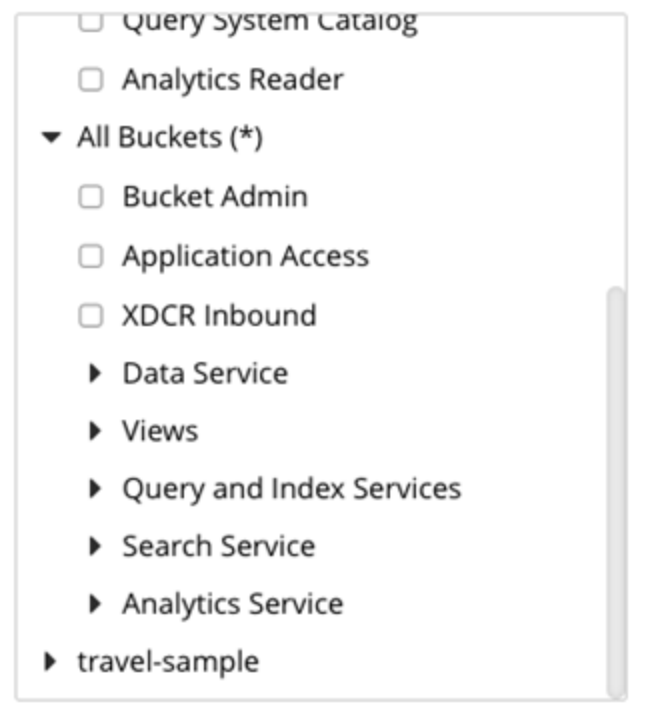

Off-the-shelf
Integration

APPDYNAMICS Prometheus Nagios DATADOG

dynatrace *And many more ...*

Centralized Security Policies

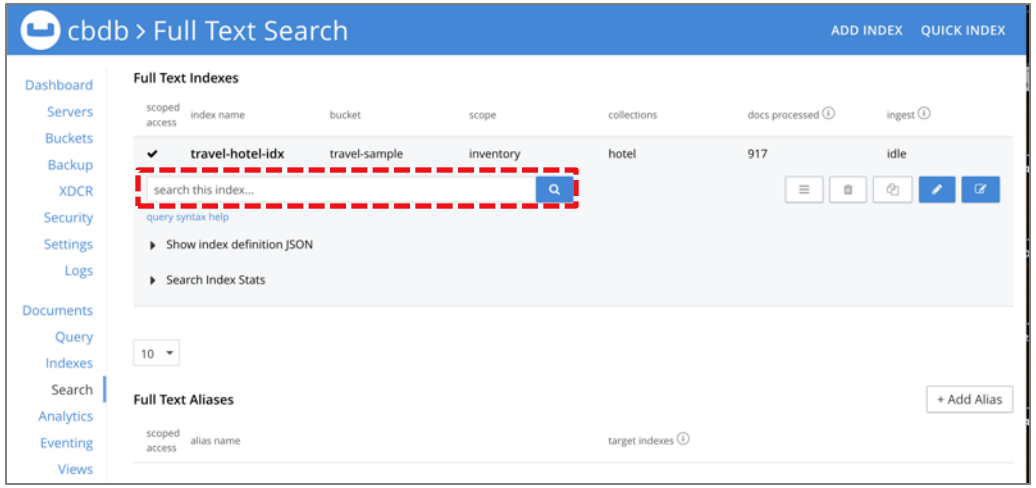
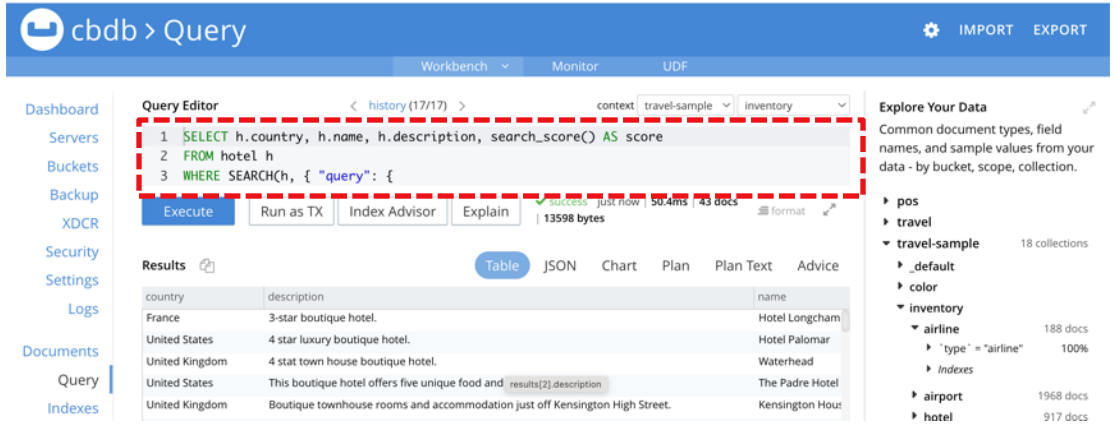
- Authorization with RBAC
- Coarse or Fine granularities
- Centralized policies for all resources, user, groups and roles



Security Policies

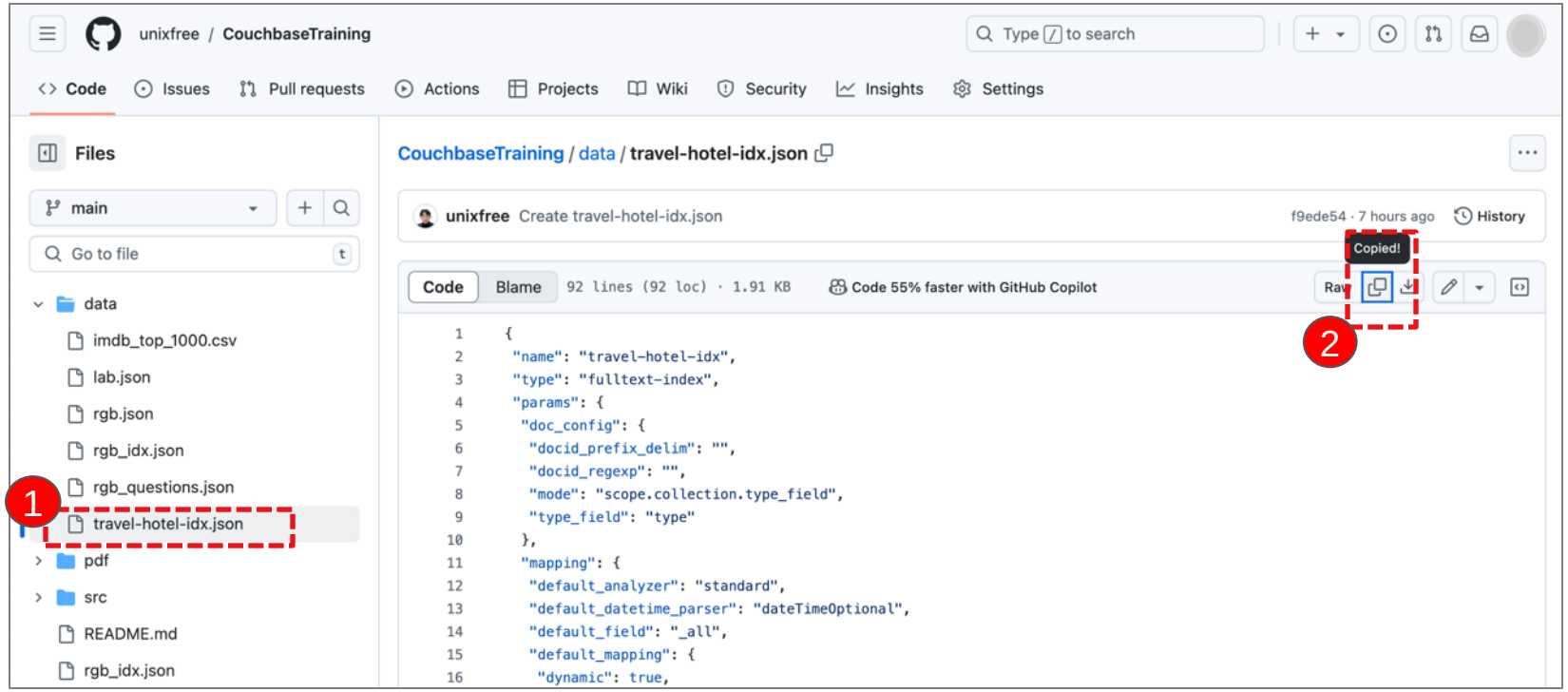
4-5. Full Text Search 실습

실습

	실습 항목	상세 실습 내용	기타
1	Index 생성	• 36페이지에서 39페이지 참고하여, 생성	
2	검색 Query 수행	• Search 페이지에서 Indeex travel-hotel-idx 선택 후, 검색 수행 • 13페이지에 있는 검색 Query 수행.	
3	SQL++ 로 검색 Query 수행	• Query 페이지에서 Context를 travel-sample > inventory 설정 • 40페이지에서 44페이지에 있는 SQL++ 검색 Query 수행.	

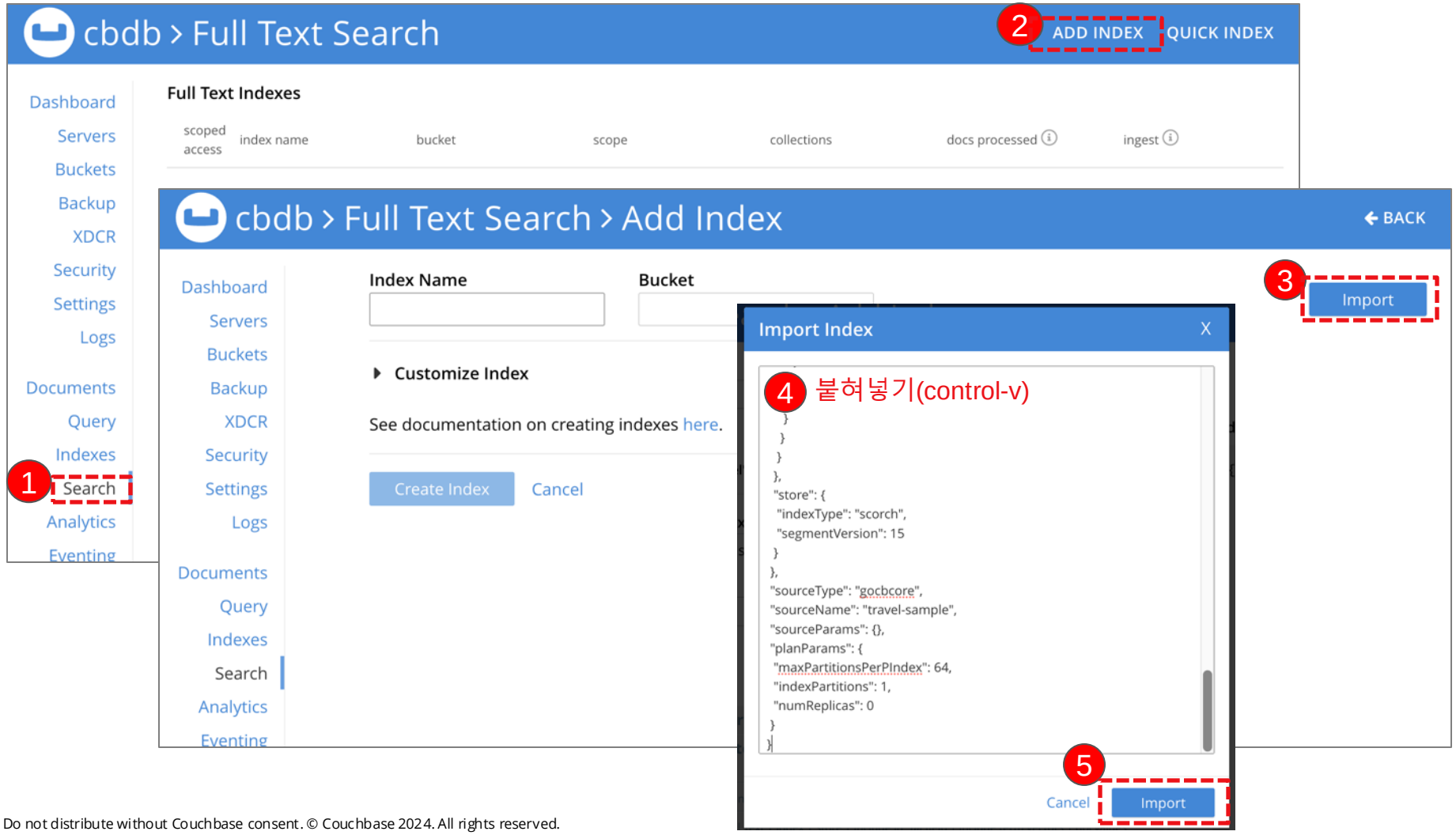
실습 > Inverted Index 생성

1. 인덱스 설정 파일 복사
   https://github.com/unixfree/CouchbaseTraining



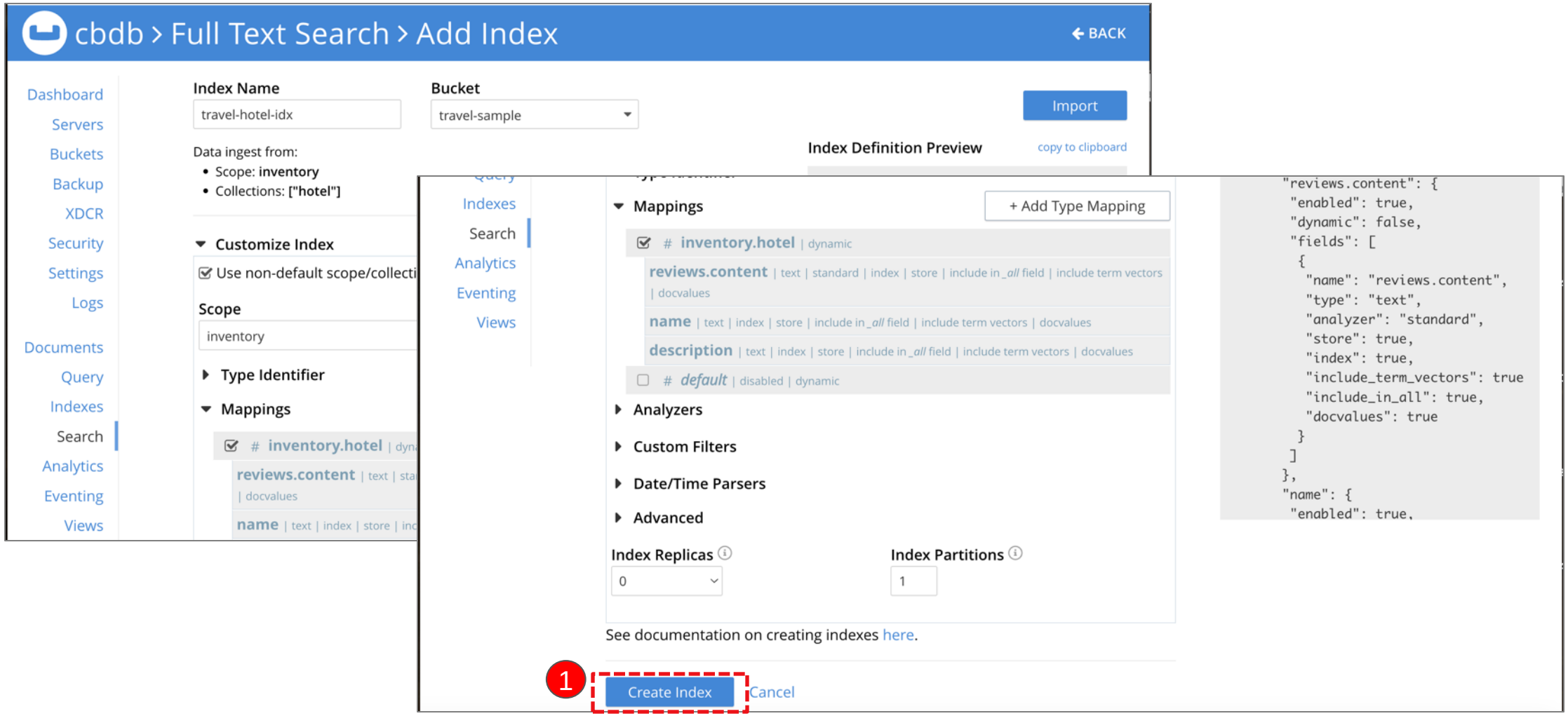
실습 > Inverted Index 생성 (계속)

2. 인덱스 설정 파일 Import로 인덱스 생성
   1) Search > 2) ADD INDEX > 3) Import > 4) 붙혀넣기(control-v) > 5) Import



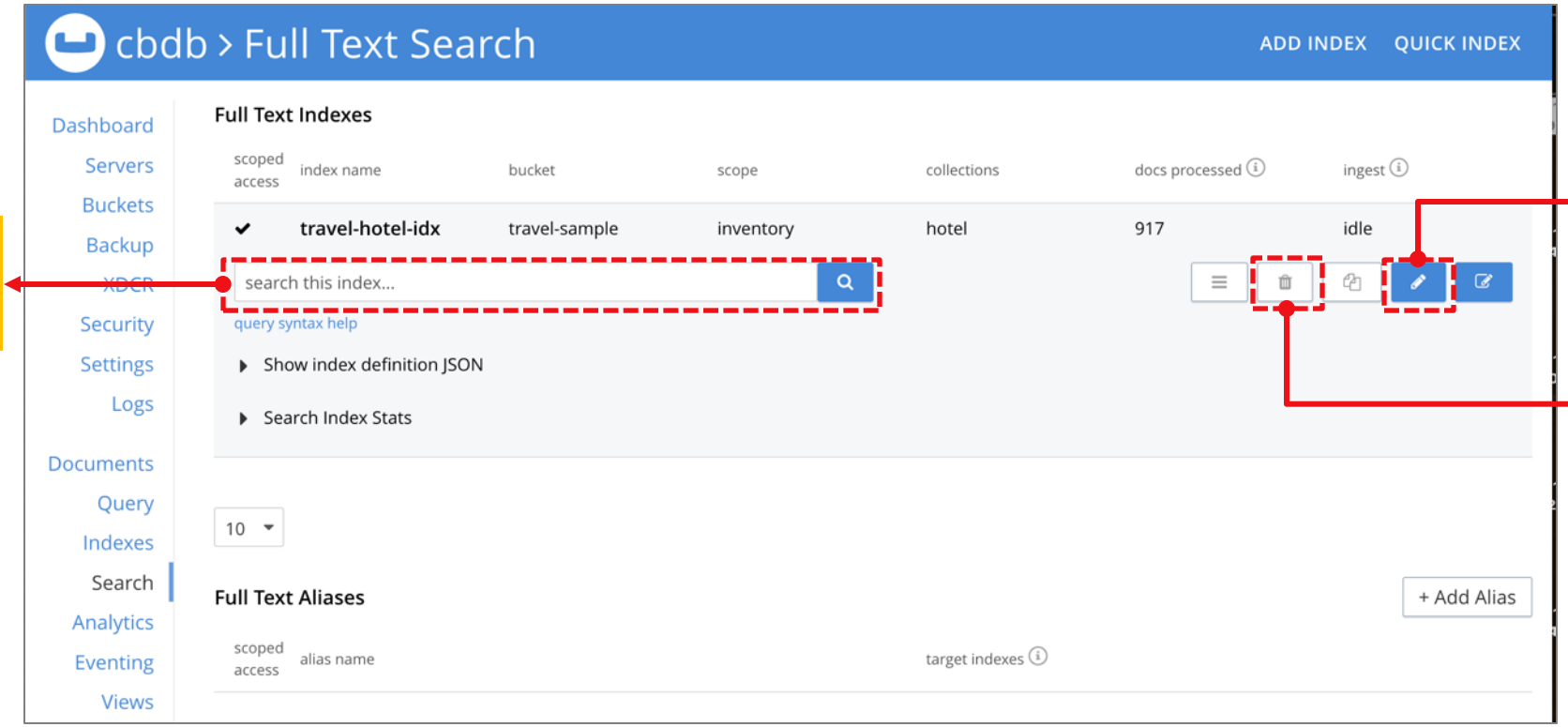
실습 > Inverted Index 생성 (계속)

2. 인덱스 설정 파일 Import로 인덱스 생성
   1) 아래와 같이 각 항목이 채워짐 > 2) 화면 스크롤 다운 후, 3) Create Index 클릭



실습 > Inverted Index 생성 (계속)

2. 인덱스 설정이 완료.
   인덱스가 만들어 지면 아래와 같이 화면(UI)가 보임.



검색어나, 검색 Query 입력창

인덱스 확인/수정

인덱스 삭제

SEARCH() : Simple Match Search

```
SELECT attribute_name
FROM collection_name
WHERE SEARCH(attribute_name, search_term)
```

```
SELECT country, name, description
FROM hotel
WHERE SEARCH(description, "boutique")
```

```
SELECT country, name, description
FROM hotel
WHERE SEARCH(description, "boutique -luxury")
```

```
SELECT country, name, description
FROM hotel
WHERE SEARCH(description, "+boutique +hotel")
```

```
SELECT country, name, description
FROM hotel
WHERE SEARCH(description, "\"boutique hotel\"")
```

SEARCH() : Simple match / match_phrase / bool Search

```
SELECT h.country, h.name, h.description, search_score() AS score
FROM hotel h
WHERE SEARCH(h, { "query": {
                              "match":"boutique",
                              "field":"description"
                            }
                 })
ORDER BY score DESC;
```

```
SELECT h.country, h.name, h.description,  search_score() AS score
FROM hotel h
WHERE SEARCH(h, { "query": {
                              "match_phrase":"boutique hotel",
                              "field":"description"
                            }
                 })
ORDER BY score DESC;
```

```
SELECT h.country, h.name, h.free_parking,  search_score() AS score
FROM hotel h
WHERE SEARCH(h, { "query": {
                              "bool":true,
                              "field":"free_parking"
                            }
                 })
ORDER BY score DESC;
```

SEARCH() : prefix / includeLocations Search

```
SELECT h.country, h.name, h.description, search_score() AS score
FROM hotel h
WHERE SEARCH(h, { "query": {
                                "prefix":"exhibit",
                                "field":"description"
                            }
                    })
ORDER BY score DESC;
```

```
SELECT h.country, h.name, h.description, SEARCH_META() as meta
FROM hotel h
WHERE SEARCH(h, { "query": {
                                "match":"boutique",
                                "field":"description"
                            },
                      "includeLocations":true
                  })
```

SEARCH() : Query Object Search – Sort & Size, AND

```
SELECT h.country, h.name, h.description, SEARCH_META() as meta
FROM hotel h
WHERE SEARCH(h, { "query": {
                                "match":"boutique",
                                "field":"description"
                            },
                    "size":5,
                    "sort":["-_score"]
                })
```

```
SELECT h.country, h.name, h.description
FROM hotel h
WHERE SEARCH(h, { "query": {
                    "conjuncts": [{
                                    "bool":true,
                                    "field":"free_parking"
                                },
                                {
                                    "bool":true,
                                    "field":"free_breakfast"
                                }]
                    }
                })
```

SEARCH() : Query Object Saerch – Fuzzy, Regular Expression

```
SELECT h.country, h.name, h.description
FROM hotel h
WHERE SEARCH(h, { "query": {
                                "match":“free wifi",
                                "field":"description",
                                "fuzziness":1
                            },
 },
                    "size":10,
                    "sort":["-_score"]
                }
            })
```

```
SELECT h.country, h.name, h.description
FROM hotel h
WHERE SEARCH(h, { "query": {
                                "regexp":"bathroom.+",
                                "field":"description"
                            }
                })
```

수고하셨습니다.

paul.son@couchbase.com

www.couchbase.com

cloud.couchbase.com

Couchbase