

## 4교시.

# Full Text Search 개요, 실습

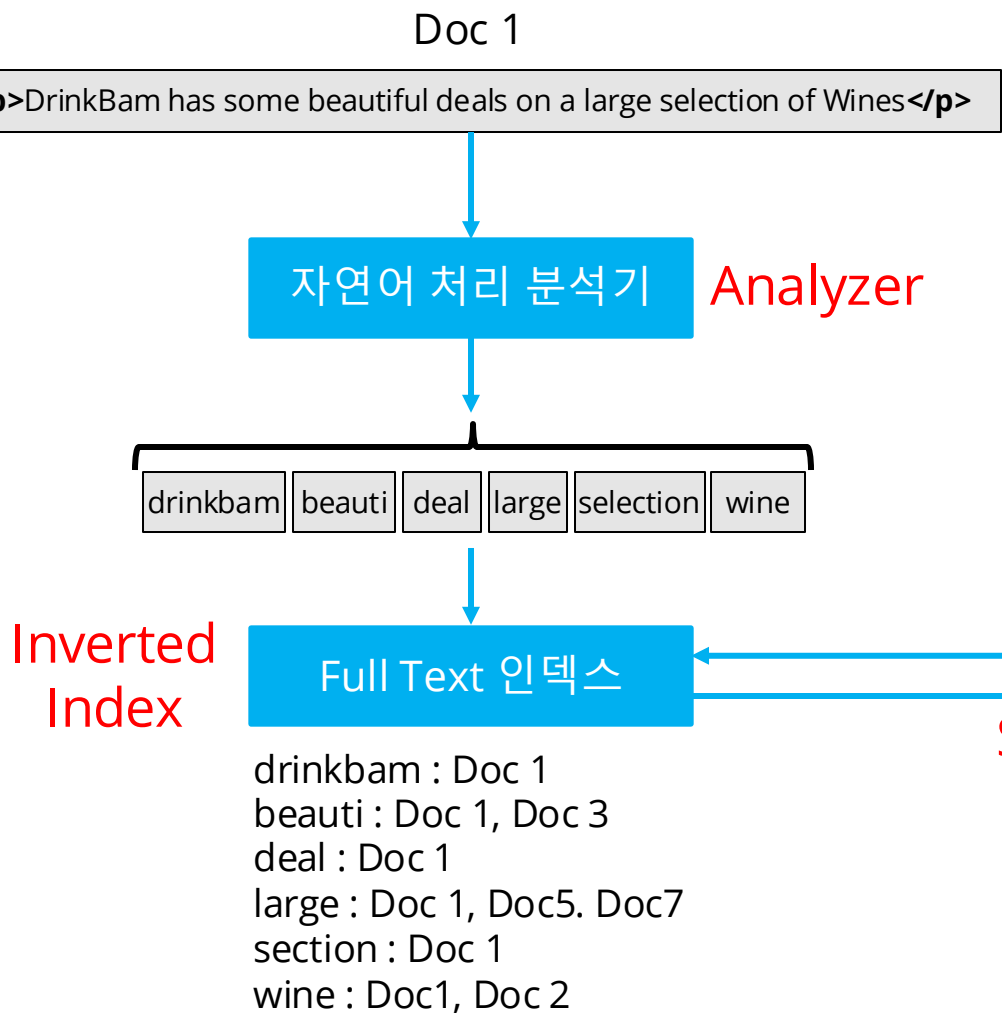
- 1 FTS 개요
- 2 Search Query
- 3 Search Results
- 4 Search Indexes
- 5 FTS 실습

## 4-1.Full Text Search 개요

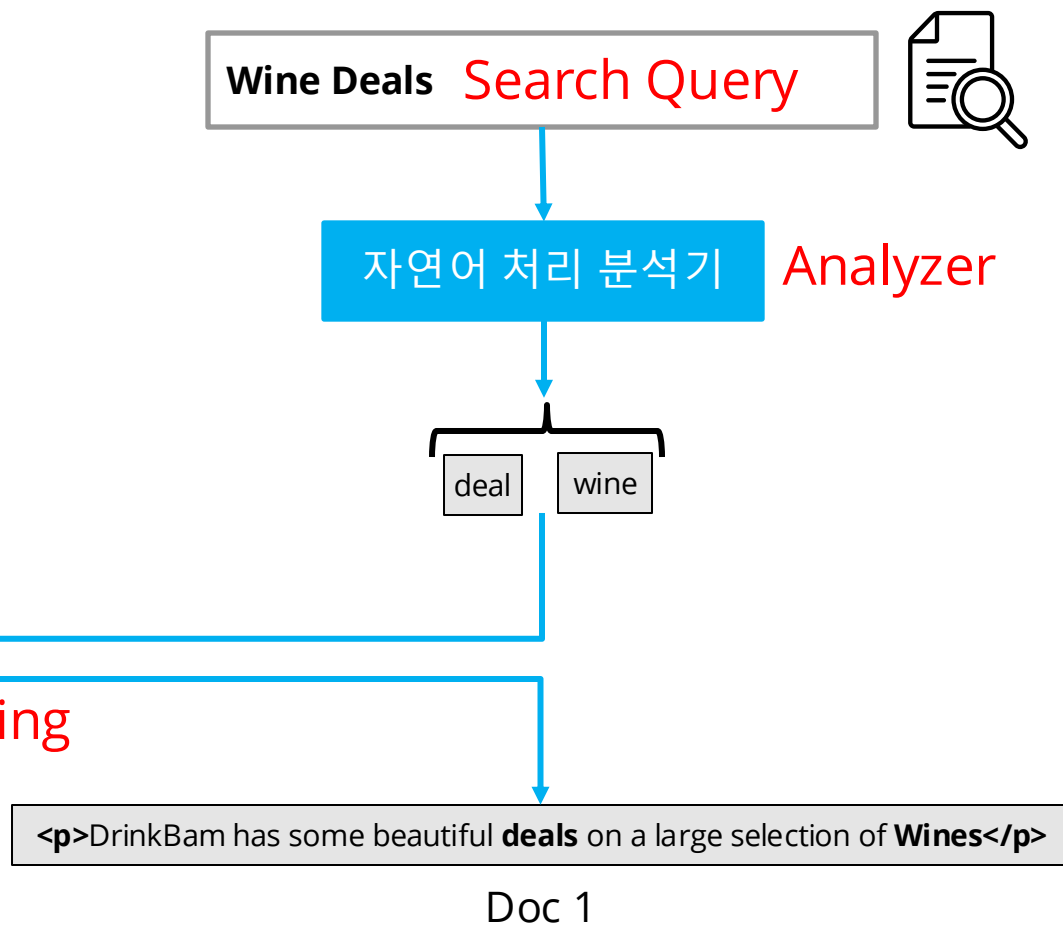


# Couchbase의 텍스트 검색 엔진

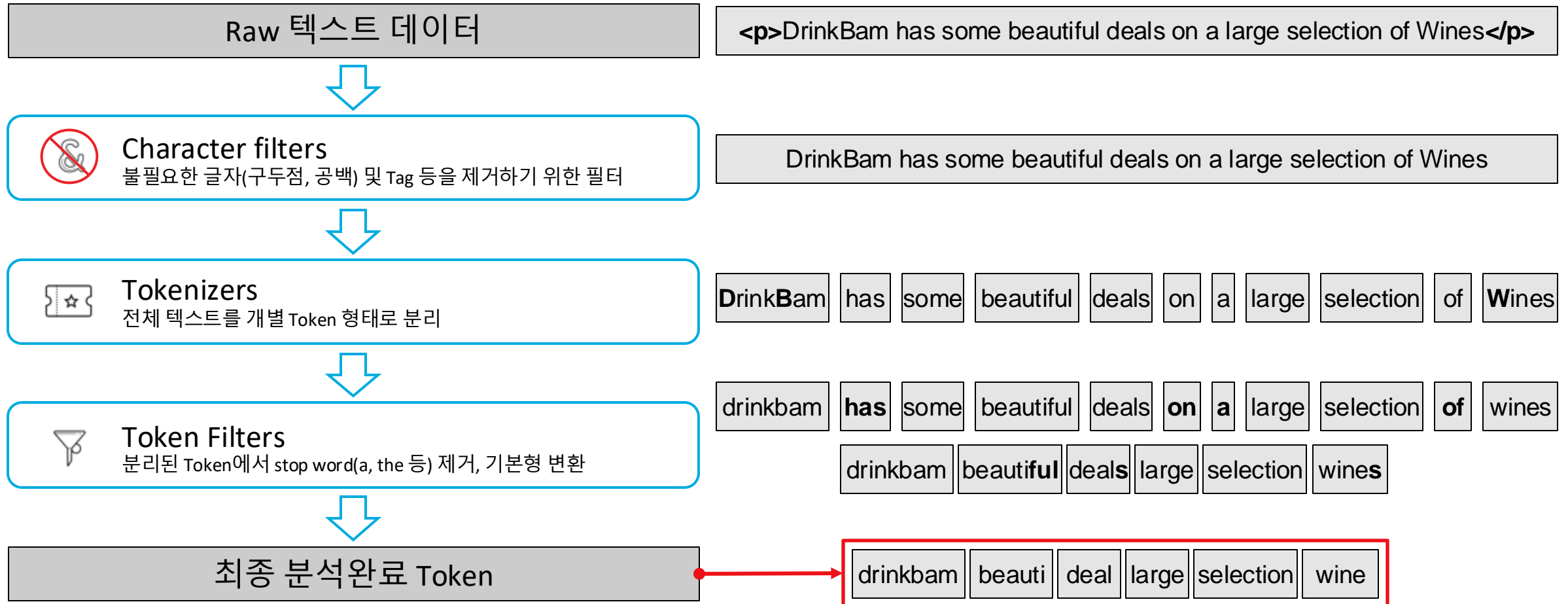
## <1. 검색을 위한 인덱스 구성>



## <2. 검색어를 통한 도큐먼트 검색>



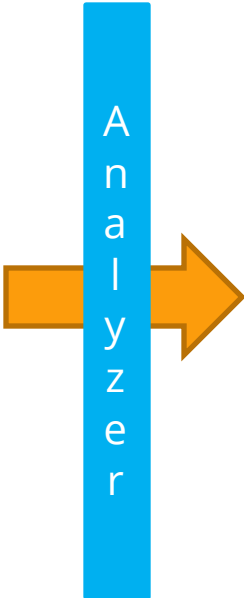
# Analyzer : 검색에 맞는 언어 식별성



# Inverted Index : 검색 성능

<Raw 텍스트 데이터>

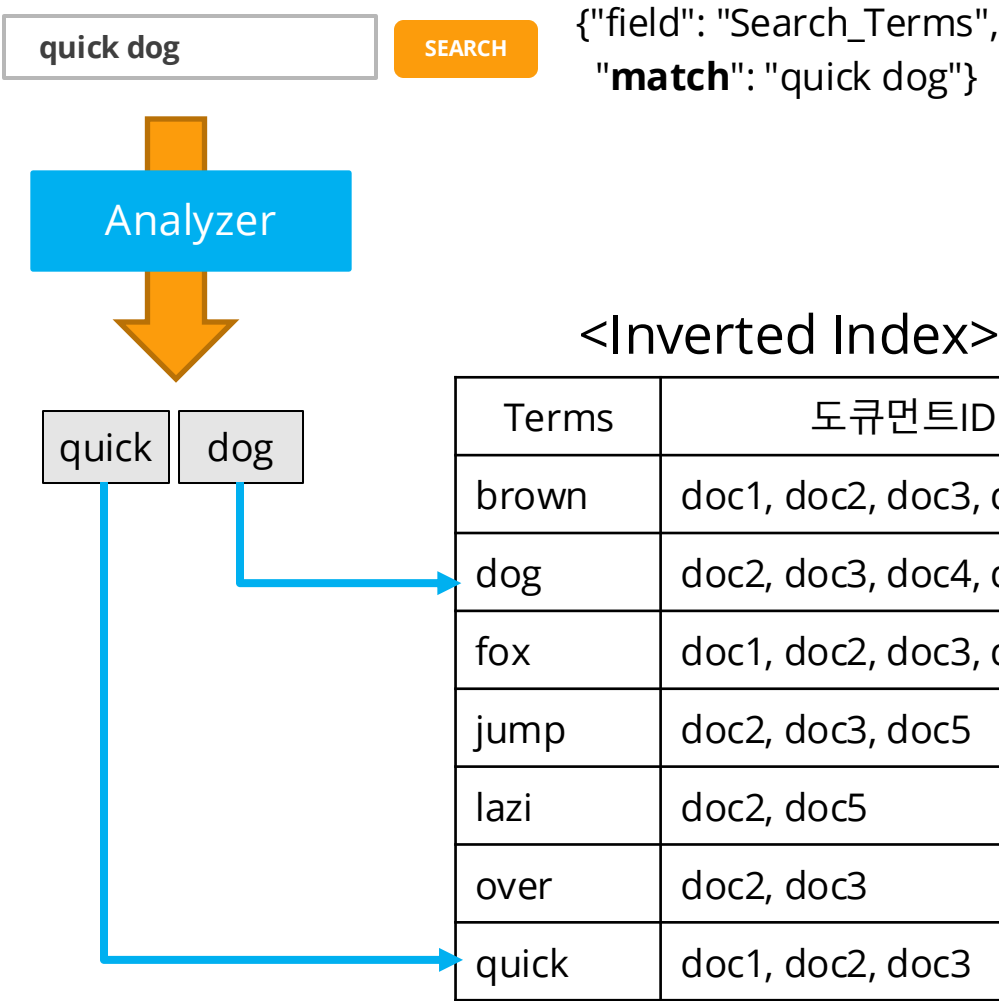
doc1	<p>The quick brown fox</p>
doc2	<p>The quick brown fox jumps over the lazy dog</p>
doc3	<p>The quick brown fox jumps over the quick dog</p>
doc4	<p>Brown fox brown dog</p>
doc5	<p>Lazy jumping dog</p>



<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



## <Search Result>

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

# Scoring : 검색 정확도

Fox are jumping

SEARCH

```
{"field": "Search_Terms",  
  "match": "Fox are jumping"}
```



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

## <Search Result>

doc3	<p>The <b>quick</b> brown fox jumps over the <b>quick dog</b> </p>	Score : 0.8762741
doc2	<p>The <b>quick</b> brown fox jumps over the lazy <b>dog</b> </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

$$\text{Score} = \text{TF} / \text{IDF}$$



## 4-2. Search Queries

>

1



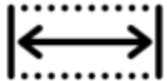
# Query types



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



SDK Clients



Through N1QL  
*Search or FLEX Indexing*

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

```
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);
}
```

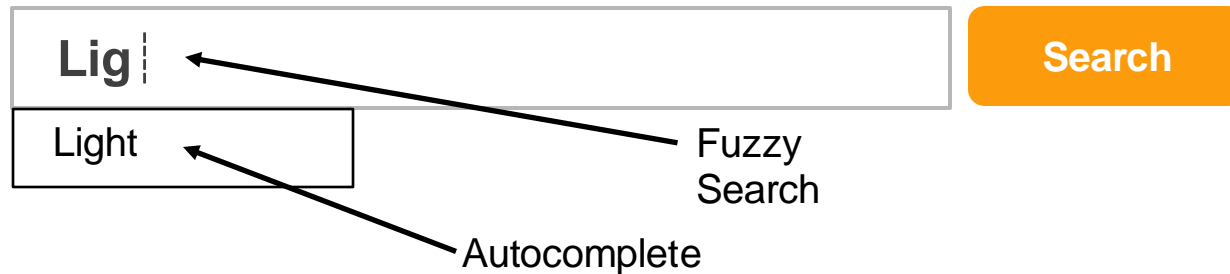
```
SELECT t1.name
FROM `travel-sample` AS t1
WHERE SEARCH(t1, {
    "match": "bathrobes",
    "field": "reviews.content",
    "analyzer": "standard"
});
```

```
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	<pre>{ "field": "reviews.content", "match": "beautiful" } { "query": { "match": "beautiful", "field": "reviews.content",   "fields": [ "reviews.content", "name" ] }</pre>	Beauty, beautiful
MATCH PHRASE	<pre>{ "field": "reviews.content", "match_phrase": "beautiful location" }</pre>	beautifully located beautiful location
FUZZY Result fields Highlight	<pre>{ "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": { } }</pre>	hotel hostel
PREFIX	<pre>{ "field": "reviews.content", "prefix": "bea" }</pre>	Beach, beautiful
REGEXP	<pre>{ "field": "reviews.content", "regexp": "ho[st t]el" }</pre>	Hostel, hotel
WILDCARD Result fields Highlight	<pre>{ "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": { } }</pre>	hostel
BOOLEAN FIELD	<pre>{ "field": "reviews.content", "bool": true, "field": "free_breakfast" }</pre>	<i>documents where the field contains boolean true value</i>

# 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.	<pre>{ "conjuncts": [   { "field": "reviews.content",     "match": "location"},   { "field": "free_breakfast",     "bool": true}] }</pre>
DISJUNCTION	Logical OR. Contains multiple child queries. Its result documents must satisfy a configurable <code>min</code> number of child queries. By default this <code>min</code> is set to 1.	<pre>{ "disjuncts": [   { "field": "reviews.content",     "match": "location"},   { "field": "free_breakfast",     "bool": true}] }</pre>
BOOLEAN	Combination of conjunction and disjunction queries and takes three lists of queries: <ul style="list-style-type: none"><li>• <code>must</code>: Result documents must satisfy all of these queries.</li><li>• <code>should</code>: Result documents should satisfy these queries.</li><li>• <code>must not</code>: Result documents must not satisfy any of these queries.</li></ul>	<pre>{ "must": {   "conjuncts": [     { "field": "reviews.content",       "match": "location"} ] },   "must_not": {     "disjuncts": [       { "field": "free_breakfast",         "bool": false } ] },   "should": {     "disjuncts": [       { "field": "free_breakfast",         "bool": true } ] } }</pre>
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.	<pre>{ "ids":   [ "hotel_10158",     "hotel_10159" ] }</pre>

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

# Query String

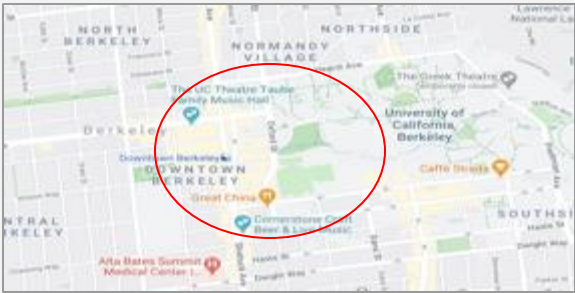
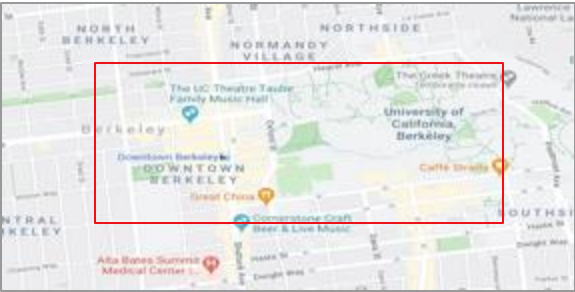
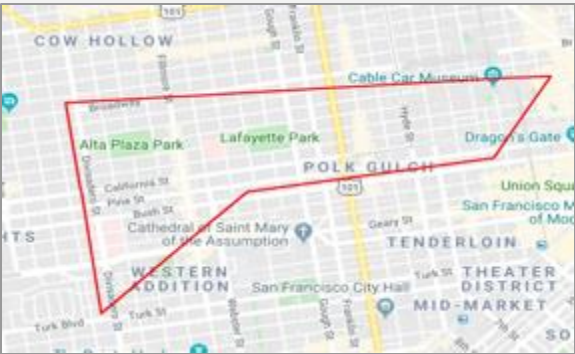
Query type	Description	Example
QUERY STRING	Query strings enable humans to describe complex queries using a simple syntax.	{ "query": "+nice +view" }

Example with Query String syntax
{ "query": "pool" }
{ "query": "continental breakfast" }
{ "query": "description:pool" }
{ "query": "+description:pool - continental breakfast" }
{ "query": "description:pool name:pool^5" }

Same example with Query syntax
{ "match": "pool", "field": "_all" }
{ "match_phrase": "continental breakfast", "field": "_all" }
{ "match": "pool", "field": "description" }
{ "must": { "conjuncts": [ { "field": "description", "match": "pool" } ], "must_not": { "disjuncts": [ { "field": "default", "match": "continental" } ], "should": { "disjuncts": [ { "field": "default", "match": "breakfast" } ] } }
<i>Boost documents having a match on name:pool</i>



# GeoSpatial Query

Query type	Description	Example
POINT DISTANCE		<pre>{   "field": "geo",   "location": "53.482358,-2.235143",   "kilometers": "1" }</pre>
BOUNDED RECTANGLE		<pre>{   "field": "geo",   "top_left": [-2.235143, 53.482358],   "bottom_right": [28.955043, 40.991862], }</pre>
BOUNDED POLYGON		<pre>{   "field": "geo",   "polygon_points": [     "37.79393211306212,-122.44234633404847",     "37.77995881733997,-122.43977141339417",     "37.788031092020155,-122.42925715405579",     "37.79026946582319,-122.41149020154114",     "37.79571192027403,-122.40735054016113" ] }</pre>



# 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'
```

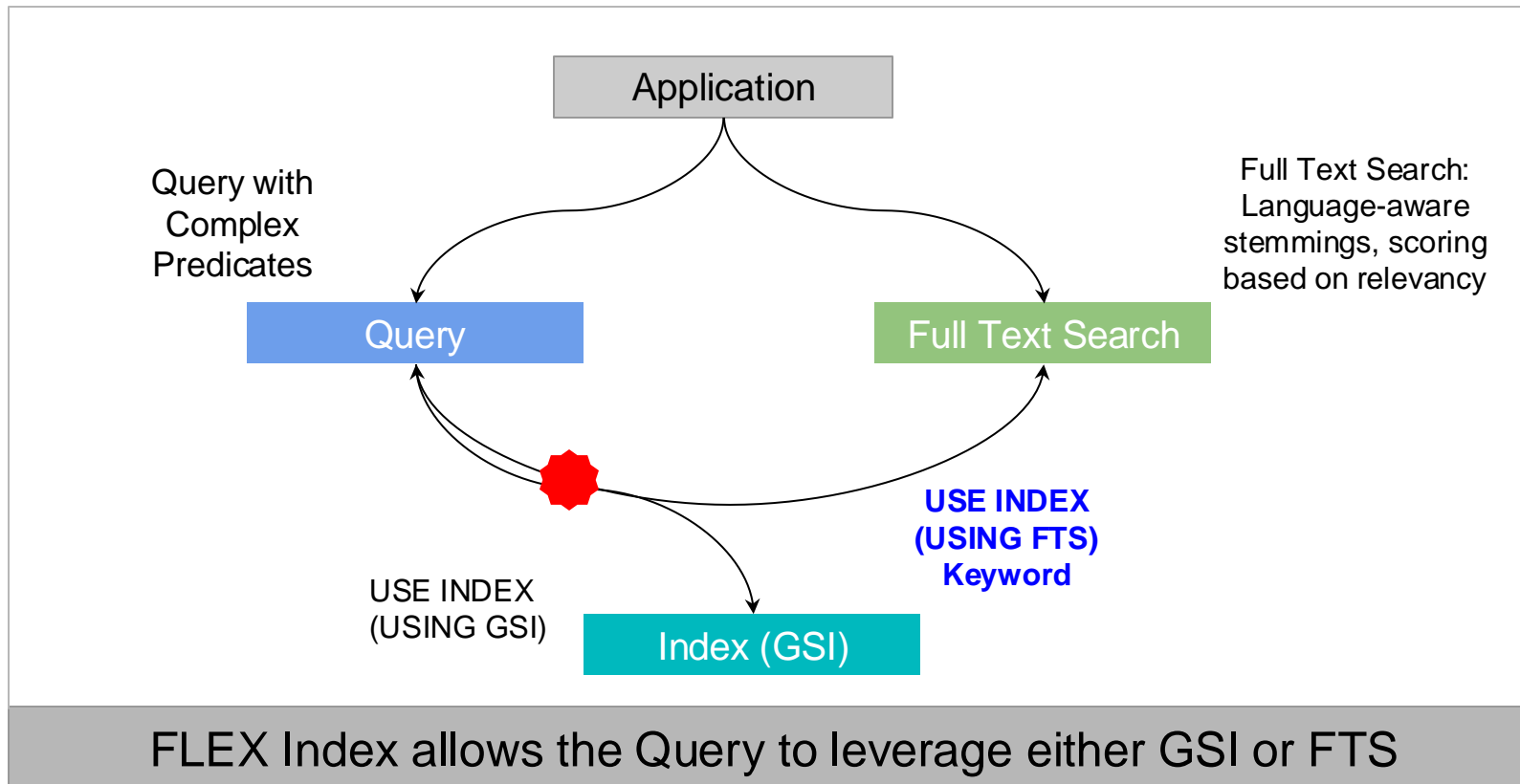
SQL clause

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



# 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 <a href="#">documentation</a> .
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 <a href="#">documentation</a> .

## 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: <ul style="list-style-type: none"><li>• field name</li><li>• <code>_id</code>:</li><li>• <code>_score</code></li></ul>	<pre>{   "fields": [ "title"],   "sort": ["country", "-_score", "-_id"],   "query":{"query": "beautiful pool"} }</pre>
Sorting with objects	Fine-grained control over sort-procedure. Each object can have the following fields: <ul style="list-style-type: none"><li>• <code>by</code></li><li>• <code>field</code></li><li>• <code>missing</code></li><li>• <code>Mode</code></li><li>• <code>type</code></li></ul>	<pre>{   ...   "sort": [     "country",     {       "by" : "field",       "field" : "reviews.ratings.Overall",       "mode" : "max",       "missing" : "last",       "type": "number"     }   ] }</pre>

# 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)	<pre>{   "query": {     "match": "California",     "field": "state"   },   "size": 5,   "from": 10 }</pre>
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.	<pre>{   "query": {     "match": "California",     "field": "state"   },   "sort": ["_id"],   "search_after": ["hotel_10180"],   "size": 3 }</pre>



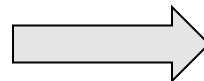
# Facets

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: [ ] Bucket: [ ]

Type Identifier

☒ JSON type field: [ type ]

☐ Doc ID up to separator: [ delimiter ]

☐ Doc ID with regex: [ regular expression ]

▼ Type Mappings [ + Add Type Mapping ]

✖ # default | dynamic

► Analyzers

► Custom Filters

► Date/Time Parsers

► Advanced

Index Replicas: [ 0 ] Index Type: [ Version 6.0 (Scorch) ] Index Partitions: [ 1 ]

[ Create Index ] [ Cancel ]

```
{
  "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

☒ # beer | only index specified fields

field

description

ok

type

text

cancel

searchable as

description

delete

analyzer

inherit

☒ index

☐ store

☒ include in \_all field

☒ include term vectors

OPTION name	Description
index	If unchecked, fields that match this will not be indexed
store	<ul style="list-style-type: none"><li>This allows the document contents to be written to the index; by default only doc IDs are written to a FTS Index</li><li>Enables highlighting and result snippets but generally results in larger indexes that are slower to build.</li><li>Encourage use of multi-gets so users don't need to store the additional information in the index.</li></ul>
Include in _all	<ul style="list-style-type: none"><li>The text in this field will be searchable in query strings without prefixing the field name.</li><li>If unchecked, the query must include this prefix (i.e. "desc:modern")</li></ul>
Include term vectors	Not storing term vectors results in smaller indexes and faster index build times.

# Search Workbench - Creating Indexes

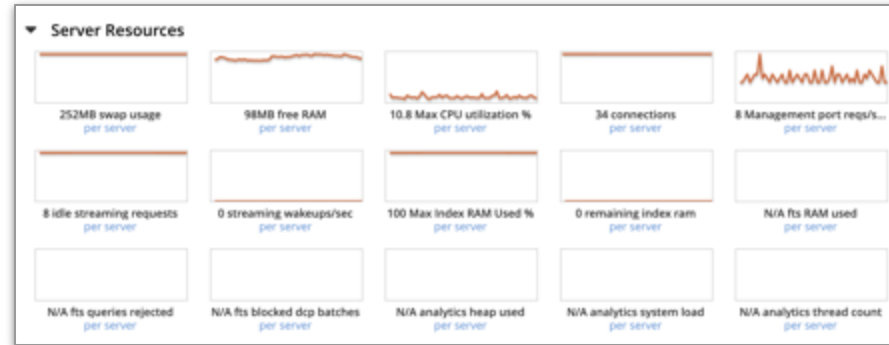
- Visual Web Console
- All features available, including Custom Analyzers creation
- Index definition available in JSON for REST API
- Once built, searchable from Web Console, SDK or REST API

The screenshot shows the 'Create Index' form in the Search Workbench. It includes fields for 'Name' and 'Bucket'. Under 'Type Identifier', there are three radio button options: 'JSON type field' (selected), 'Doc ID up to separator', and 'Doc ID with regex'. Each has a corresponding text input field with placeholder text 'type', 'delimiter', and 'regular expression' respectively. A 'Type Mappings' section shows a list with a checked item '# default | dynamic' and a '+ Add Type Mapping' button. Below this are expandable sections for 'Analyzers', 'Custom Filters', 'Date/Time Parsers', and 'Advanced'. At the bottom, there are fields for 'Index Replicas' (set to 0), 'Index Type' (set to 'Version 6.0 (Scorch)'), and 'Index Partitions' (empty). The form concludes with 'Create Index' and 'Cancel' buttons.

# Monitoring Search



Couchbase  
Web Console



RESTful  
API

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



Off-the-shelf  
Integration



**Nagios**

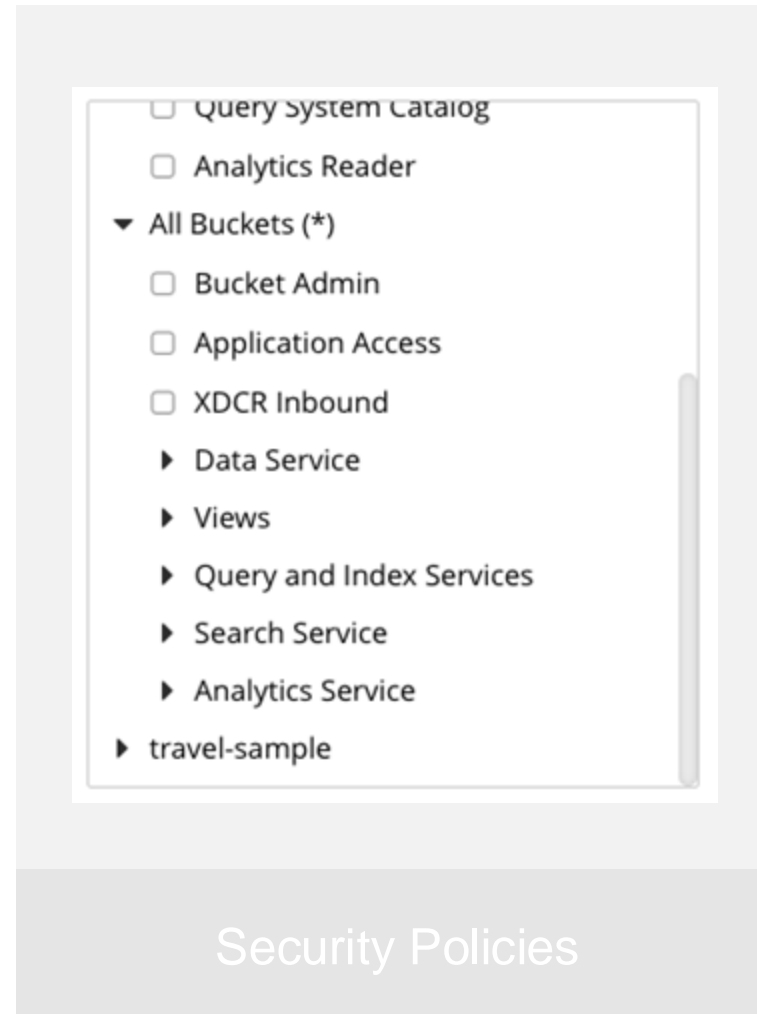


*And many more ...*



# Centralized Security Policies

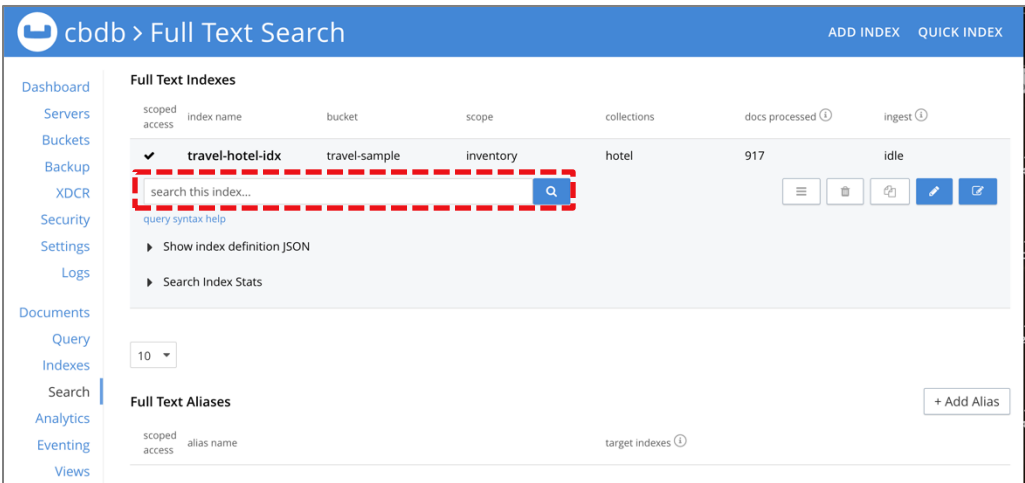
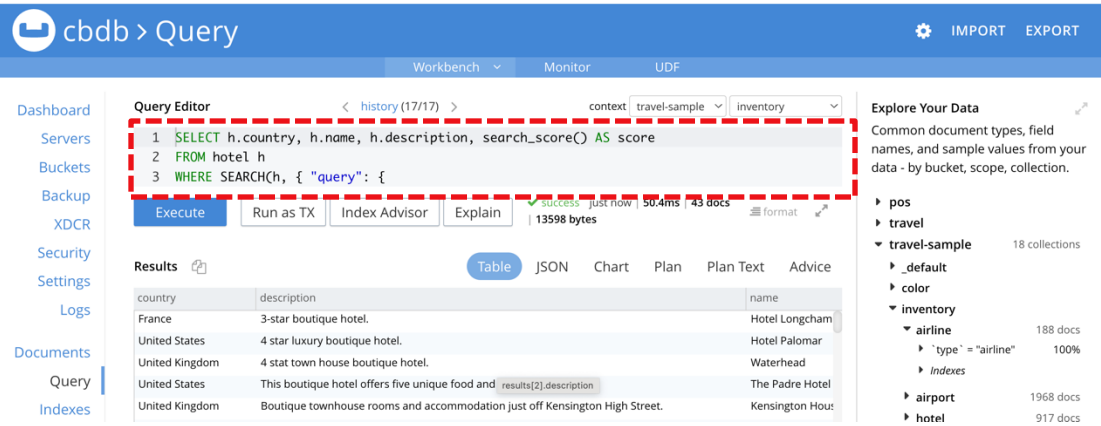
- Authorization with RBAC
- Coarse or Fine granularities
- Centralized policies for all resources, user, groups and roles



## 4-5. Full Text Search 실습



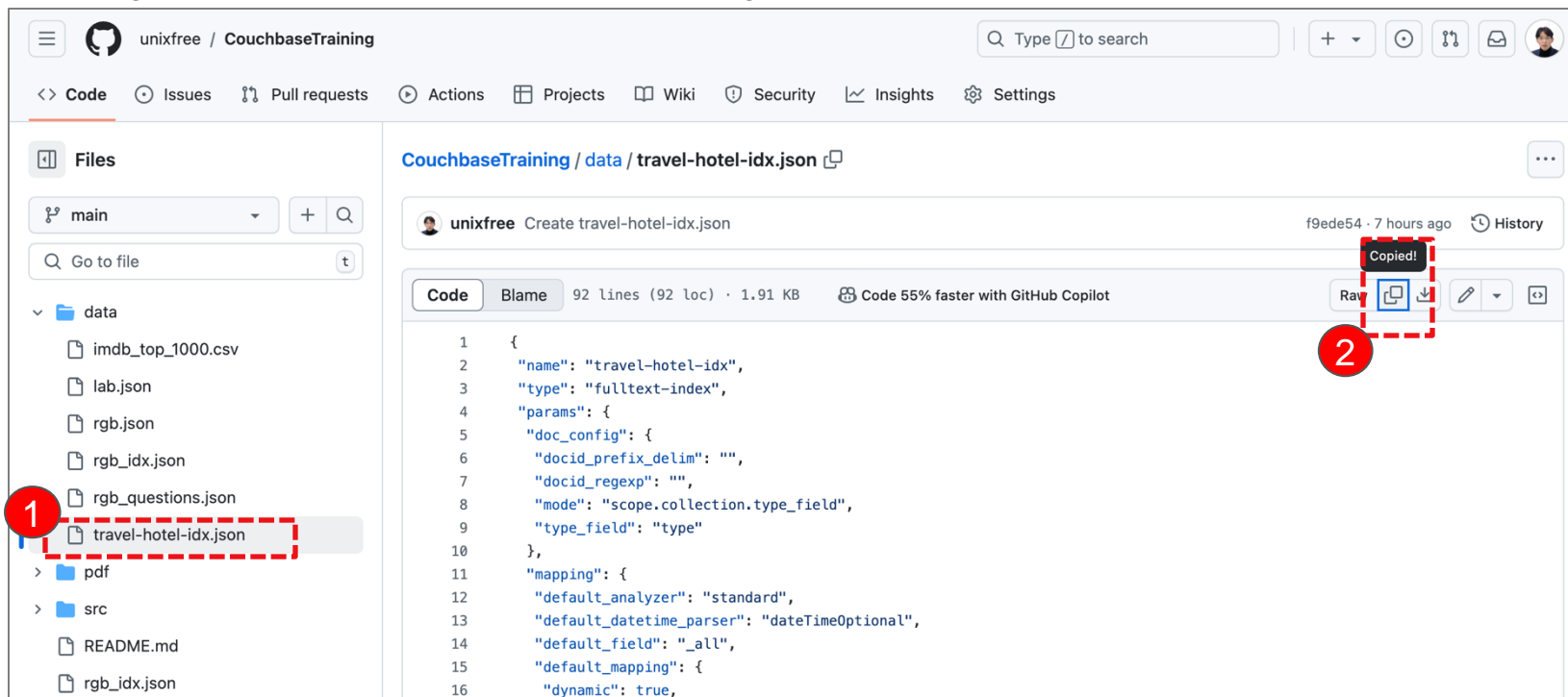
# 실습

	실습 항목	상세 실습 내용	기타
1	Index 생성	<ul style="list-style-type: none"> <li>36페이지에서 39페이지 참고하여, 생성</li> </ul>	
2	검색 Query 수행	<ul style="list-style-type: none"> <li>Search 페이지에서 Index <b>travel-hotel-idx</b> 선택 후, 검색 수행</li> <li>13페이지에 있는 검색 Query 수행.</li> </ul>	
3	SQL++ 로 검색 Query 수행	<ul style="list-style-type: none"> <li>Query 페이지에서 Context를 <b>travel-sample &gt; inventory</b> 설정</li> <li>40페이지에서 44페이지에 있는 SQL++ 검색 Query 수행.</li> </ul>	

# 실습 > Inverted Index 생성

## 1. 인덱스 설정 파일 복사

<https://github.com/unixfree/CouchbaseTraining>



The screenshot shows the GitHub interface for the repository 'unixfree / CouchbaseTraining'. The left sidebar displays the file tree with the 'data' directory expanded, showing files like 'imdb\_top1000.csv', 'lab.json', 'rgb.json', 'rgb\_idx.json', 'rgb\_questions.json', and 'travel-hotel-idx.json'. A red dashed box labeled '1' highlights 'travel-hotel-idx.json'. The main area shows the content of 'travel-hotel-idx.json', which is a JSON configuration for a Couchbase index. A red dashed box labeled '2' highlights the copy icon in the file viewer toolbar. The JSON content is as follows:

```
1 {
2   "name": "travel-hotel-idx",
3   "type": "fulltext-index",
4   "params": {
5     "doc_config": {
6       "docid_prefix_delim": "",
7       "docid_regexp": "",
8       "mode": "scope.collection.type_field",
9       "type_field": "type"
10    },
11    "mapping": {
12      "default_analyzer": "standard",
13      "default_datetime_parser": "dateTimeOptional",
14      "default_field": "_all",
15      "default_mapping": {
16        "dynamic": true,
```

# 실습 > Inverted Index 생성 (계속)

## 2. 인덱스 설정 파일 Import로 인덱스 생성

1) Search > 2) ADD INDEX > 3) Import > 4) 붙여넣기(control-v) > 5) Import

The screenshot illustrates the process of creating an inverted index in the Couchbase Search console. It is divided into three main sections:

- Top Section (Full Text Search):** The header bar shows 'cbdb > Full Text Search'. A red circle with the number '2' highlights the 'ADD INDEX' button in the top right corner.
- Middle Section (Add Index):** The main content area is titled 'cbdb > Full Text Search > Add Index'. It features a sidebar on the left with navigation links: Dashboard, Servers, Buckets, Backup, XDCR, Security, Settings, Logs, Documents, Query, Indexes, Search (highlighted with a red circle and number '1'), Analytics, and Eventing. The main area has fields for 'Index Name' and 'Bucket'. Below these is a 'Customize Index' section with a link to documentation and 'Create Index' and 'Cancel' buttons. A red circle with the number '3' highlights the 'Import' button in the top right corner.
- Bottom Section (Import Index Modal):** A modal window titled 'Import Index' is open. It contains a text area with a JSON configuration for an inverted index. A red circle with the number '4' highlights the text area, with the instruction '붙여넣기(control-v)' (Paste with control-v) next to it. At the bottom of the modal, a red circle with the number '5' highlights the 'Import' button, with a 'Cancel' button next to it.

```
{
  "store": {
    "indexType": "scorch",
    "segmentVersion": 15
  },
  "sourceType": "gocbcore",
  "sourceName": "travel-sample",
  "sourceParams": {},
  "planParams": {
    "maxPartitionsPerIndex": 64,
    "indexPartitions": 1,
    "numReplicas": 0
  }
}
```

# 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

cbdb > Full Text Search > Add Index

Dashboard  
Servers  
Buckets  
Backup  
XDCR  
Security  
Settings  
Logs  
Documents  
Query  
Indexes  
Search  
Analytics  
Eventing  
Views

Index Name  
travel-hotel-idx

Bucket  
travel-sample

Import

Data ingest from:

- Scope: inventory
- Collections: ["hotel"]

Customize Index

Use non-default scope/collection

Scope  
inventory

Type Identifier

Mappings

# inventory.hotel | dynamic

reviews.content | text | standard | index | store | include in \_all field | include term vectors | docvalues

name | text | index | store | include in \_all field | include term vectors | docvalues

description | text | index | store | include in \_all field | include term vectors | docvalues

# default | disabled | dynamic

Index Definition Preview

copy to clipboard

Mappings

+ Add Type Mapping

# inventory.hotel | dynamic

reviews.content | text | standard | index | store | include in \_all field | include term vectors | docvalues

name | text | index | store | include in \_all field | include term vectors | docvalues

description | text | index | store | include in \_all field | include term vectors | docvalues

# default | disabled | dynamic

Analyzers

Custom Filters

Date/Time Parsers

Advanced

Index Replicas ⓘ  
0

Index Partitions ⓘ  
1

See documentation on creating indexes [here](#).

1

Create Index

Cancel

```
"reviews.content": {
  "enabled": true,
  "dynamic": false,
  "fields": [
    {
      "name": "reviews.content",
      "type": "text",
      "analyzer": "standard",
      "store": true,
      "index": true,
      "include_term_vectors": true,
      "include_in_all": true,
      "docvalues": true
    }
  ]
},
{
  "name": {
    "enabled": true,
```

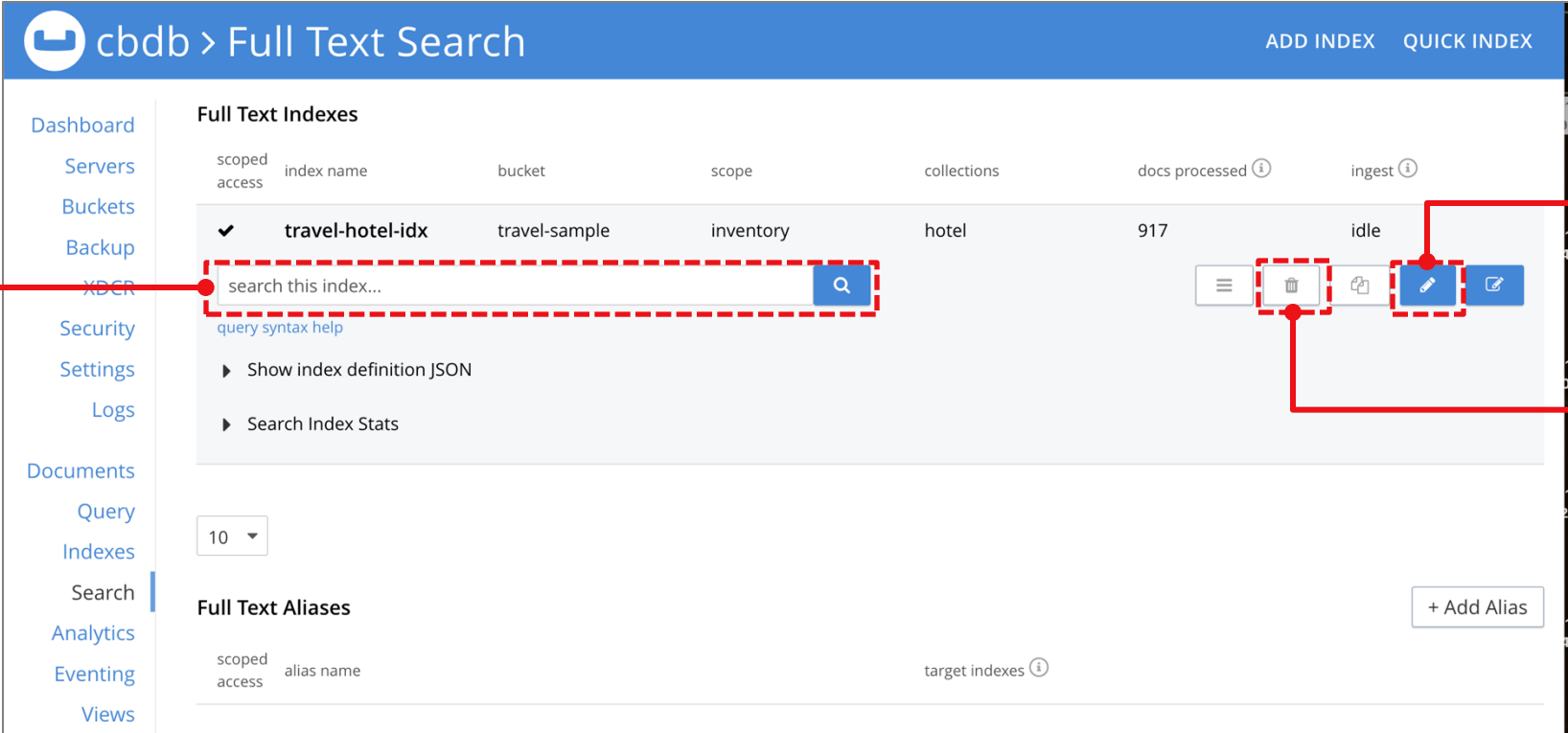
Confidential and Proprietary. Do not distribute without Couchbase consent. © Couchbase 2024. All rights reserved.

38

# 실습 > Inverted Index 생성 (계속)

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

검색어나, 검색  
Query 입력창



The screenshot shows the Couchbase Full Text Search interface. The top navigation bar includes 'cbdb > Full Text Search' and links for 'ADD INDEX' and 'QUICK INDEX'. A left sidebar lists various database management options. The main content area displays a table of 'Full Text Indexes' with columns for 'scoped access', 'index name', 'bucket', 'scope', 'collections', 'docs processed', and 'ingest'. One index, 'travel-hotel-idx', is highlighted. Below the table, there is a search input field with a magnifying glass icon, a 'query syntax help' link, and links for 'Show index definition JSON' and 'Search Index Stats'. To the right of the index row are several action buttons: a menu icon, a delete icon, a copy icon, an edit icon, and a share icon. Below the table, there is a section for 'Full Text Aliases' with columns for 'scoped access', 'alias name', and 'target indexes', and a '+ Add Alias' button.

scoped access	index name	bucket	scope	collections	docs processed	ingest
✓	travel-hotel-idx	travel-sample	inventory	hotel	917	idle

search this index... [magnifying glass icon]

query syntax help

► Show index definition JSON

► Search Index Stats

[menu icon] [delete icon] [copy icon] [edit icon] [share icon]

+ Add Alias

scoped access	alias name	target indexes
---------------	------------	----------------

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 Search – 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"
  },
  "size": 10,
  "sort": [ "_score" ]
})
```



# 수고하셨습니다.



[paul.son@couchbase.com](mailto:paul.son@couchbase.com)

[www.couchbase.com](http://www.couchbase.com)

[cloud.couchbase.com](http://cloud.couchbase.com)



**Couchbase**