Oracle NoSQL Database

Full Text Search

12c Release 1

(Library Version 12.1.4.0)

ORACLE
NOSQL DATABASE

Legal Notice

6/3/2016

Table of Contents

Introduction

Full Text Search (or just text search) provides the capability to identify natural-language documents that satisfy a query, and optionally to sort them by relevance to the query. The most common type of search is to find all documents containing given query terms and return them in order of their similarity to the query. Notions of query and similarity are very flexible and depend on the specific application. The simplest search considers query as a set of words and similarity as the frequency of query words in the document.

Oracle NoSQL Database integrates with a third-party open-source search engine, Elasticsearch (ES) to enable text-searching capability in Oracle NoSQL Database, in-concert with the Tables interface. Full-text search is an important aspect of any big data and Oracle NoSQL Database system. Users expect that when they input text into a box and click "search", they will get the relevant search results they are looking for in a fraction of a second. This feature provides:

- High performance full-text search of Tables stored in Oracle NoSQL Database
- Search which allows users to explore a collection of information by applying multiple filters

Note

So that the maintenance of indexes does not affect the performance of an Oracle NoSQL Database store, text indexes will not be maintained locally by Oracle NoSQL Database components, but will instead be maintained by a remote service (Elasticsearch) hosted on other nodes.

This feature provides a means of marking fields in an Oracle NoSQL Database Tables schema as being text searchable. Text indexing allows creating indexes on Oracle NoSQL Database tables that cause the indexed fields to automatically enter into an Elasticsearch cluster. Once the data is in Elasticsearch, you may use any native Elasticsearch API to search and retrieve it. The documents retrieved from Elasticsearch contain references back to the original Oracle NoSQL Database records, facilitating their retrieval.

Note

This is a preview release of Full Text Search. As a preview release its use and feedback is encouraged. For more details on what it means for a feature to be a preview release feature, please see the release notes.

Note

Oracle strongly recommends that you do not use the Full Text Search feature with a secure Oracle NoSQL Database installation.

Prerequisite

To use this feature, you must have an Elasticsearch 2.0 cluster running as well as the Oracle NoSQL Database store. In a production environment, for performance reason, both Oracle NoSQL Database nodes and Elasticsearch nodes are intended to be used in distributed environments with different hosts.

- You can download ES here:

  https://www.elastic.co/downloads/past-releases/elasticsearch-2-0-0
- You can find the installation instructions here:

  https://www.elastic.co/guide/en/elasticsearch/reference/2.0/_installation.html

When your ES cluster is running, it will consist of one or more nodes. Some or all of the nodes will have services listening on two ports.

- The HTTP port, which is used for REST requests. It is 9200 by default.
- The ES transport port, which is used for communication between ES nodes. It is 9300 by default.

Note

You must know the host name and transport port of at least one node in the cluster, and the name of the cluster itself, which by default is "elasticsearch". See the command `show parameters` in Integrating Elasticsearch with Oracle NoSQL Database (page 4). This information will be provided to the Oracle NoSQL Database store so that it can connect to the ES cluster.

Integrating Elasticsearch with Oracle NoSQL Database

Before you can create a text index, you must register the Elasticsearch cluster with the Oracle NoSQL Database store, using the `register-es` plan command. In this command you provide the ES cluster name, and the host name and transport port of any node in the cluster as follows:

```
plan register-es -clustername <name> -host <host>
-port <transport port> [-force]
```

For example:

```
kv-> plan register-es -clustername elasticsearch
-host 127.0.0.1 -port 9300
Started plan 5. Use show plan -id 5 to check status.
To wait for completion, use plan wait -id 5
```

Note

You will see an error message if the Elasticsearch cluster already contains "stale indexes" corresponding to the Oracle NoSQL Database. A stale index is one that was not created by the current instance of the store, but by a previous instance of the store, or by a concurrent instance of a store with the same name as the current store, which is not allowed.

The optional `-force` argument causes the Oracle NoSQL Database store to initialize an Elasticsearch cluster regardless of whether it already contains a stale index corresponding to the Oracle NoSQL Database store. See for example:

```
kv-> plan register-es -clustername elasticsearch
-host localhost -port 9300 -force
```

```
Started plan 38. Use show plan -id 38 to check status.
To wait for completion, use plan wait -id 38
```

Oracle NoSQL Database store Admin communicates with the Elasticsearch node to verify its existence, and it will acquire a complete list of connection information for all the nodes in the ES cluster. This information will be stored and distributed to all the nodes of the Oracle NoSQL Database store. This command can be repeated if the Elasticsearch cluster's population of nodes changes significantly, to update Oracle NoSQL Database's list of Elasticsearch node connections.

If you want to verify that your KV is registered with the ES, run the following command:

```
show parameters -service <storage node id>
```

This has a list of properties which also includes the cluster instance and the name of the cluster. See the `searchClusterMembers=127.0.0.1:9300` and `searchClusterName=elasticsearch` in the output below:

```
kv-> show parameters -service sn1
capacity=1
haHostname=localhost
haPortRange=5005,5007
hostname=localhost
memoryMB=0
mgmtClass=oracle.kv.impl.mgmt.NoOpAgent
mgmtPollPort=0
mgmtTrapPort=0
numCPUs=8
registryPort=5000
rnHeapMaxMB=0
rnHeapPercent=85
rootDirPath=./kvroot
searchClusterMembers=127.0.0.1:9300
searchClusterName=elasticsearch
serviceLogFileCount=20
serviceLogFileLimit=2000000
storageNodeId=1
systemPercent=10
```

To deregister an Elasticsearch cluster from the Oracle NoSQL Database store, use the following command:

```
kv-> plan deregister-es
Executed plan 16, waiting for completion...
Plan 16 ended successfully
```

This is allowed only if all full text indexes are first removed using the following command:

```
DROP INDEX [IF EXISTS] index_name ON table_name
```

For more information, see Drop Index (page 13). Otherwise, you get the following error message:

```
kv-> plan deregister-es
Cannot deregister ES because these text indexes exist:
mytestIndex
```

```
JokeIndex
```

To verify if the deregistration of the ES was successful or not, run the `show parameters -service <storage node id>` command.

Creating Full Text Index

You can create text indexes by using this DDL command:

```
CREATE FULLTEXT INDEX [if not exists] <index-name> ON <table-name>
    (<field-name> [ <mapping-spec> ], ...)
    [ES_SHARDS = <n>] [ES_REPLICAS = <n>]
```

For more information, see Example - Creating Full Text Search (page 8).

After you create the text index, you can verify the same by using the following statement:

```
show indexes - table <name>
```

After you create the index, you can run the `show table` command that lists the full text index that you have created. This command will give the table structure including the indexes that have been created for that table:

```
show table -name <tableName>
```

For example:

```
kv-> show table -name mytestTable
{
"type" : "table",
"name" : "mytestTable",
"owner" : null,
"comment" : null,
"shardKey" : [ "id" ],
"primaryKey" : [ "id" ],
"fields" : [ {
"name" : "id",
"type" : "INTEGER",
"nullable" : true,
"default" : null
}, {
"name" : "category",
"type" : "STRING",
"nullable" : true,
"default" : null
}, {
"name" : "txt",
"type" : "STRING",
"nullable" : true,
"default" : null
} ],
"indexes" : [ {
"name" : "mytestIndex",
"comment" : null,
"fields" : [ "category", "txt" ]
} ]
}
```

Note

You cannot evolve an index. If you want to change the index definition, for example, add more columns to the index structure, you have to delete the index and create a new one.

Mapping in Elasticsearch

This DDL command is similar to the command that creates regular secondary indexes. One difference is the addition of the optional <mapping-spec> clause that follows the field name. If present, <mapping-spec> is a small JSON document that influences Elasticsearch's treatment of the field. The other difference is optional settings of shards and replicas for the ES index.

When a user creates a text index, an ES index will be created and named as ondb.<kvstore>.<table>.<textIndex>. For more information, see the section Creating ES Index for Each Text Index in KVStore (page 8).

The index contains a single mapping which is generated from the CREATE FULLTEXT INDEX command. See, https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html#mapping-type

One aspect of the mapping is a list of fields that compose the document type, along with their types. In the absence of a <mapping-spec>, Oracle NoSQL Database will supply a default type for each field that corresponds to the type of the column in the Oracle NoSQL Database table. For example, if a table column A has the type string, then the mapping supplied to Elasticsearch will declare a field named A of type string. If you want Elasticsearch to treat column A as an integer despite its being a string in Oracle NoSQL Database, you must provide an explicit type by including a <mapping-spec> clause:

```
{ "type" : "integer" }
```

The <mapping-spec>, in addition to specifying the type of the field, can also contain any of a large set of parameters for determining how Elasticsearch handles the field. For example, if you want to store the field, but not index it (that is, not make it available for search), you would include the tuple "index" : "no". For information on a list of such parameters, see: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html

You may supply a <mapping-spec> that does not include the `type` key, and Oracle NoSQL Database will supply the default type.

The following scalar column type will be mapped to ES, STRING, INTEGER, LONG, BOOLEAN, FLOAT, DOUBLE. For example a Long ("LONG") in Oracle NoSQL Database will be mapped to Long ("long") in ES.

Note

Indexed fields can include non-scalar types, which are specified in the same way and with the same limitations as those for secondary indexes. For more information, see Indexing Non-Scalar Data Types in *Oracle Getting Started with Oracle NoSQL Database Tables*.

Creating ES Index for Each Text Index in KVStore

For each text index in kvstore, an ES index is created with a unique name:

```
ondb.<kvstore>.<table>.<textIndex>
```

For example, for a text index `mytestIndex` in table `mytestTable` in store mystore (mystore should be set in a fixed width font, as are mytestIndex and mytestTable), the corresponding ES index would be: *ondb.mystore.mytesttable.mytestindex*.

```
ondb.<kvstore>.<parentTable>.<childTable>.<textIndex>
```

Since the ES index name uniquely identifies the text index in kvstore, the ES index type will be constant name as *text_index_mapping*.

Example - Creating Full Text Search

1. Create a table as follows:

```
kv-> execute 'CREATE TABLE mytestTable
(id INTEGER, category STRING, txt STRING, PRIMARY KEY (id))'
Statement completed successfully
```

2. Use the following command to make a text index on that table that indexes the category and txt columns:

```
kv-> execute 'CREATE FULLTEXT INDEX mytestIndex
ON mytestTable (category, txt)'
Statement completed successfully
```

3. Insert data into the "mytestTable" Table:

```
kv-> put table -name mytestTable -json
'{ "id" : 1, "category" : "pun", "txt" : "Spring is natures way of
saying, Let us party" }'
Operation successful, row inserted.
```

```
kv-> put table -name mytestTable -json
'{ "id" : 2, "category" : "self-referential", "txt" : "I am thankful
for the mess to clean after a party because it means I have been
surrounded by friends" }'
Operation successful, row inserted.
```

```
kv-> put table -name mytestTable -json
'{ "id" : 3, "category" : "stupid", "txt" : "Doing nothing is hard,
you never know when you are done" }'
Operation successful, row inserted.
```

```
kv-> put table -name mytestTable -json
'{ "id" : 4, "category" : "thoughtful", "txt" : "Do not worry if plan
A fails, there are 25 more letters in the alphabet" }'
Operation successful, row inserted.
```

```
kv-> get table -name mytestTable
{"id":4,"category":"thoughtful","txt":"Do not worry if plan A fails,
there are 25 more letters in the alphabet"}
```

```
{"id":1,"category":"pun","txt":"Spring is natures way of saying,
Let us party"}
{"id":2,"category":"self-referential","txt":"I am thankful for the
mess to clean after a party because it means I have been surrounded
by friends"}
{"id":3,"category":"stupid","txt":"Doing nothing is hard, you never
know when you are done"}
4 rows returned
```

Note

As you enter these records, Oracle NoSQL Database produces a document that is sent to Elasticsearch for indexing. You can find the document by searching the Elasticsearch cluster.

elasticsearch-head is a plugin of Elasticsearch. For more information, see https://mobz.github.io/elasticsearch-head/. It is recommended to use one of the Elasticsearch plugins to identify if the documents are getting indexed.

Note

Once you get the result back from ES, you can use list of _id returned by elasticsearch to retrieve the records from Oracle NoSQL Database.

4. Search Elasticsearch cluster to find the document that was created in the steps before. ES allows REST calls as queries, so we can do REST calls using the `cURL` command. The `cURL` command sends an http request to the ES node listening on port 9200.

Note

`cURL` is a common utility program that can issue and display the results of http requests. Currently, it is supported on Microsoft Windows, Linux, and Mac OS X. For more information, see: https://en.wikipedia.org/wiki/CURL and Search Using `cURL` command (page 9). However, `cURL` is an alternative method/option for querying Elasticsearch. The other options can be using:

- elasticsearch-head, a web front end for browsing and interacting with an Elasticsearch cluster, helps to query. elasticsearch-head is part of ES standard installation and can be enabled by following the steps mentioned here: https://mobz.github.io/elasticsearch-head/

  For more information, see Search using elasticsearch-head (page 11).

- Java API commands, see section Search Text Index using JAVA APIs (page 12).

Search Using `cURL` command

- Use the following cURL command to produce every document that is indexed with the mytestIndex mapping:

```
curl -s localhost:9200/ondb.mystore
.mytesttable.mytestindex/_search\?pretty
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 4,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0003",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"3"},"category":
      "stupid","txt":       "Doing nothing is hard, you never know when
      you are done"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0002",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"2"},"category":
      "self-referential","txt": "I am thankful for the mess to clean
      after a party because it means I have been surrounded by friends"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0004",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"4"},"category":
      "thoughtful","txt": "Do not worry if plan A fails, there are 25
      more letters in the alphabet"}
    }, {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0001",
      "_score" : 1.0,
      "_source":{"_pkey":{"_table":"mytestTable","id":"1"},"category":
      "pun","txt": "Spring is natures way of saying, Let us party"}
    } ]
  }
}
```

Note

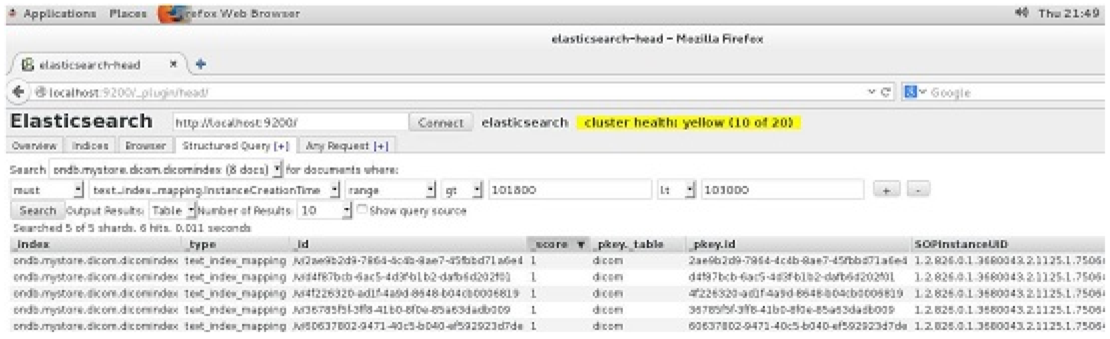
- The name that we gave to the text index in the CREATE FULLTEXT statement was "mytestIndex", so we are restricting the search to the ES index associated with that Oracle NoSQL Database text index.
- The "cURL" command above asked to search for every record in mytestIndex (there is no search term, so every record matches the search). The argument "pretty" means to pretty-print the output.
- The result contains an array of "hits" with a single member. The interesting property is "_source" which contains "_pkey" which has the table and primary key for the original kvstore record; and the two indexed fields "category" and "txt".
- Each item has a "_score" field which Elasticsearch uses to indicate the level of relevance for search hits. For more information, see: https://www.elastic.co/guide/en/elasticsearch/guide/current/relevance-intro.html

- You can narrow the search by putting a search term into request, using "q=" like the example below:

```
curl -s localhost:9200/ondb.mystore.mytesttable
.mytestindex/_search\?q=25\&pretty
{
  "took" : 11,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.25,
    "hits" : [ {
      "_index" : "ondb.mystore.mytesttable.mytestindex",
      "_type" : "text_index_mapping",
      "_id" : "/w/¨0004",
      "_score" : 0.25,
      "_source":{"_pkey":{"_table":"mytestTable","id":"4"},"category":
      "thoughtful","txt": "Do not worry if plan A fails, there are 25
      more letters in the alphabet"}
    } ]
  }
}
```

Search using elasticsearch-head

An aggregated query looks as follows:


Applications Places Firefox Web Browser Thu 21:49
elasticsearch-head - Mozilla Firefox
elasticsearch-head
localhost:9200/_plugin/head/
Google
Elasticsearch
Connect
elasticsearch
cluster health: yellow (10 of 20)
Overview
Indices
Browser
Structured Query [+]
Any Request [+]
Search
Show query source

Search ondb.mystore.dicom.dicomindex (8 docs) for documents where:

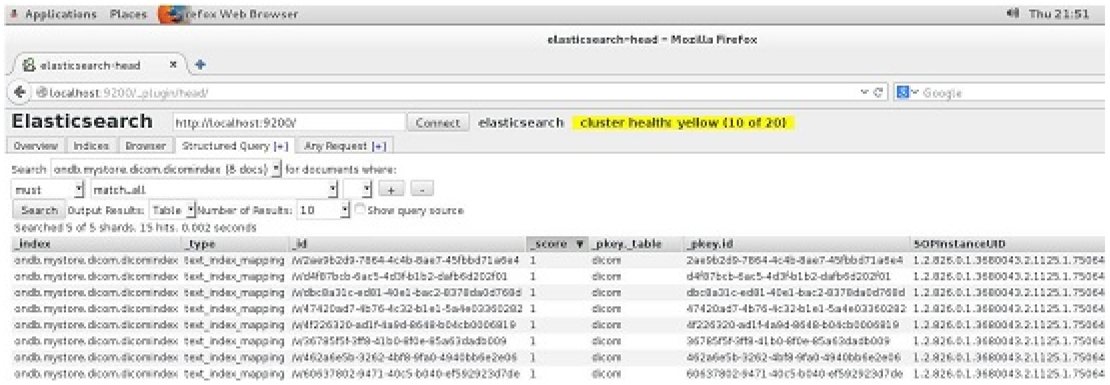
must | text_index_mapping.InstanceCreationTime | range | gt | 101800 | lt | 103000

Output Results: Table Number of Results: 10

Searched 5 of 5 shards. 6 hits. 0.011 seconds

_index	_type	_id	_score	_pkey._table	_pkey.id	SOPInstanceUID
ondb.mystore.dicom.dicomindex	text_index_mapping	/w2ae9b2d9-7864-4c4b-8ae7-45fbbd71a6e4	1	dicom	2ae9b2d9-7864-4c4b-8ae7-45fbbd71a6e4	1.2.826.0.1.3680043.2.1125.1.7506
ondb.mystore.dicom.dicomindex	text_index_mapping	/wd4f87bcb-6ac5-4d3f-b1b2-dafb6d202f01	1	dicom	d4f87bcb-6ac5-4d3f-b1b2-dafb6d202f01	1.2.826.0.1.3680043.2.1125.1.7506
ondb.mystore.dicom.dicomindex	text_index_mapping	/w4f226320-ad1f-4a9d-8648-b04cb0006819	1	dicom	4f226320-ad1f-4a9d-8648-b04cb0006819	1.2.826.0.1.3680043.2.1125.1.7506
ondb.mystore.dicom.dicomindex	text_index_mapping	/w36785f5f-3ff8-41b0-8f0e-85a63dadb009	1	dicom	36785f5f-3ff8-41b0-8f0e-85a63dadb009	1.2.826.0.1.3680043.2.1125.1.7506
ondb.mystore.dicom.dicomindex	text_index_mapping	/w60637802-9471-40c5-b040-ef592923d7de	1	dicom	60637802-9471-40c5-b040-ef592923d7de	1.2.826.0.1.3680043.2.1125.1.7506

A query with "match_all" looks as follows:


Applications Places Firefox Web Browser Thu 21:51
elasticsearch-head - Mozilla Firefox
elasticsearch-head
localhost:9200/_plugin/head/
Google
Elasticsearch
Connect
elasticsearch
cluster health: yellow (10 of 20)
Overview
Indices
Browser
Structured Query [+]
Any Request [+]
Search
Show query source

Search ondb.mystore.dicom.dicomindex (8 docs) for documents where:

must | match_all

Output Results: Table Number of Results: 10

Searched 5 of 5 shards. 15 hits. 0.002 seconds

_index	_type	_id	_score	_pkey._table	_pkey.id	SOPInstanceUID
ondb.mystore.dicom.dicomindex	text_index_mapping	/w2ae9b2d9-7864-4c4b-8ae7-45fbbd71a6e4	1	dicom	2ae9b2d9-7864-4c4b-8ae7-45fbbd71a6e4	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/wd4f87bcb-6ac5-4d3f-b1b2-dafb6d202f01	1	dicom	d4f87bcb-6ac5-4d3f-b1b2-dafb6d202f01	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/wdbc8a31c-ed81-40e1-bac2-8378da0d768d	1	dicom	dbc8a31c-ed81-40e1-bac2-8378da0d768d	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/w47420ad7-4b76-4c32-b1e1-5a4e03360282	1	dicom	47420ad7-4b76-4c32-b1e1-5a4e03360282	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/w4f226320-ad1f-4a9d-8648-b04cb0006819	1	dicom	4f226320-ad1f-4a9d-8648-b04cb0006819	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/w36785f5f-3ff8-41b0-8f0e-85a63dadb009	1	dicom	36785f5f-3ff8-41b0-8f0e-85a63dadb009	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/w462a6e5b-3262-4bf8-9fa0-4940bb6e2e06	1	dicom	462a6e5b-3262-4bf8-9fa0-4940bb6e2e06	1.2.826.0.1.3680043.2.1125.1.75064
ondb.mystore.dicom.dicomindex	text_index_mapping	/w60637802-9471-40c5-b040-ef592923d7de	1	dicom	60637802-9471-40c5-b040-ef592923d7de	1.2.826.0.1.3680043.2.1125.1.75064

For more information, see https://mobz.github.io/elasticsearch-head/

Search Text Index using JAVA APIs

For information on creating Oracle NoSQL Database tables, see Introducing Oracle NoSQL Database Tables and Indexes in the *Getting Started with Oracle NoSQL Database Tables* guide. To create a text index, you must use the method KVStore.executeSync to issue the DDL command CREATE FULLTEXT INDEX, as mentioned in the section Example - Creating Full Text Search (page 8).

Searching the index, on the other hand, must be done using Elasticsearch APIs. Here is a very simple example of a program that searches a document type that corresponds to an Oracle NoSQL Database text index. This command, given the arguments "localhost 9300 kvstore MyIndex 25" produces exactly the same output as the curl command in the section Search Using cURL command (page 9).

Note

To build and run this program, you will need all jar files supplied with the Elasticsearch distribution in your class path. One way to achieve this would be to use the java command's class path wildcard feature, for example:

```
java -cp ".:/home/…/elasticsearch-2.0.0/lib/*" \
    DoSearch localhost 9300 kvstore mytestIndex 25
```

See the following example program:

```
import java.net.InetAddress;

import org.elasticsearch.action.search.SearchRequestBuilder;
import org.elasticsearch.action.search.SearchResponse;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.transport.InetSocketTransportAddress;
import org.elasticsearch.index.query.QueryBuilders;

public class DoSearch {

    public static void main(String args[]) throws Exception {

        if (args.length < 4 || args.length > 5) {
            System.err.println
                ("Usage: DoSearch <esTransportHost> <esTransportPort> " +
                 "<kvStoreName> <tableName> <indexName> [search-term]");
        }
        String esTransportHost = args[0];
        int esTransportPort = Integer.parseInt(args[1]);
        String kvStoreName = args[2];
        String tableName = args[3];
        String indexName = args[4];
        String searchTerm = args.length > 5 ? args[5] : null;

        TransportClient client =
            TransportClient.builder().build();

        client.addTransportAddress
            (new InetSocketTransportAddress
             (InetAddress.getByName(esTransportHost), esTransportPort));

        final String esIndexName = "ondb." + kvStoreName.toLowerCase()
        + "." +
            tableName + "." + indexName;

        SearchRequestBuilder sb = client.prepareSearch(esIndexName);
        if (searchTerm != null) {
            sb.setQuery(QueryBuilders.simpleQueryStringQuery(searchTerm));
        }

        SearchResponse response = sb.execute().actionGet();
        System.out.println(response);
    }
}
```

Drop Index

Text indexes share the same namespace as regular secondary indexes within a table. You can use the same statement to remove either type of index. DROP INDEX on a text index stops the population of the index from Oracle NoSQL Database shards, and removes the mapping and all related documents from ES. If a fulltext index is to be modified, then it must first be undeployed, and then re-deployed after modification. There is no means of "evolving" a fulltext index.

If a table to which a text index mapping refers is dropped, the text index will automatically be dropped as part of the process.

You can drop text indexes by using this DDL command:

```
DROP INDEX [IF EXISTS] index_name ON table_name
DROP TABLE [IF EXISTS] table_name
```

For example:

```
kv-> execute 'drop table mytestTable'
Statement completed successfully
```

Troubleshooting

The most common problems that might arise when using Oracle NoSQL Database with Elasticsearch are those related to data transfer failure and data not getting indexed. For information on troubleshooting in ES, see: https://www.elastic.co/guide/en/elasticsearch/hadoop/current/troubleshooting.html

The following sections describe some of the causes of these issues and provides steps you can follow to resolve these problems. Here are some things you can check to verify whether the data is successfully transferred and indexed:

- You can verify Oracle NoSQL Database's information that it uses to connect to the Elasticsearch cluster by issuing the runadmin command `show parameters -service sn1` where `sn1` is the id of any storage node in the store. The parameters of interest are `searchClusterName` which is the Elasticsearch cluster name; and `searchClusterMembers` which is a list of `host:port` representing the nodes in the Elasticsearch cluster.
- Be sure that when you are registering the ES cluster, you give an ES node's transport port and not its http port.
- You can do a quick check of connectivity from the Oracle NoSQL Database master administrative node by re-registering the ES cluster using the command `plan register-es`. This plan is safe to run multiple times. If it runs without errors, then the ES cluster is at least available to the administrative node.
- Both Oracle NoSQL Database and ES nodes should be configured to listen on appropriate network interfaces. If an Oracle NoSQL Database store is using a public interface, but Elasticsearch is using the loopback interface, then they will not be able to communicate properly. The network interface is configured for an Elasticsearch node by the property `network.host` in the `elasticsearch.yml`, and for an Oracle NoSQL Database storage node by the `-host` option to the `makebootconfig` command.
- You can issue a command like `curl http://localhost:9200/_cat/indices` to get a list of the indexes in an Elasticsearch cluster, to verify that they correspond to the text

indexes you created in Oracle NoSQL Database. The output of this command will also show an indication of the status of each index using the color names green, yellow, and red. If the status is red, then the index cannot be populated.

- If you see unexplained failures to index, see the RepNode logs for SEVERE log messages or exceptions related to Elasticsearch. For information about finding logs, see Software Monitoring in the *Oracle NoSQL Database Runbook*.

- With a heavily update-dominated work load, the Elasticsearch cluster can lag quite far behind Oracle NoSQL Database. It can take minutes for a new Oracle NoSQL Database record to be reflected in search results from Elasticsearch. This issue can be mitigated by increasing the number of Elasticsearch nodes, tuning Elasticsearch, and especially by storing Elasticsearch data on solid state disks.

- Compare Record counts

  You can compare the number of records in table for Oracle NoSQL Database with the number of documents in your Elasticsearch cluster (this assumes that the Oracle NoSQL Database has a static number of items). This is particularly useful, for instance your cluster is in a test environment where the number of records is set and you do not add more. To find the number of records in Oracle NoSQL Database, use the following statement in the Command Line Interface (CLI):

  ```
  kv-> aggregate table -name mytestTable -count
  Row count: 100
  ```

  To get the number of records for Elasticsearch, use the following command: http://localhost:9200/ondb.mystore.joke.jokeindex/_count

  If it is successful, you get the following response:

  ```
  {"count":100,"_shards":
      {
      "total":5,"successful":5,"failed":0
      }
  }
  ```

  The `count` returned by Elasticsearch is the same value as the number of records shown in the CLI. The matching values provide assurance that all records have been transferred and indexed by Elasticsearch.

- ElasticServer Version Mismatch

  You must use Elasticsearch 2.0 version. A different version, for example ES 1.7.3, populates the following error:

  ```
  kv-> plan register-es -clustername elasticsearch -host
  localhost -port 9300
  Can't connect to an Elasticsearch cluster at localhost:9300
  {elasticsearch}
  ```

  The Admin log shows this:

```
2016-04-05 20:20:19.512 UTC INFO
[admin1] [Washout] loaded [], sites []
2016-04-05 20:20:20.434 UTC INFO [admin1] [Washout]
failed to get local cluster state for
{#transport#-1}{127.0.0.1}{localhost/127.0.0.1:9300},
disconnecting...
RemoteTransportException[[Failed to deserialize response
of type [org.elasticsearch.action.admin.cluster.state.
ClusterStateResponse]]];
nested: TransportSerializationException
[Failed to deserialize response of
type [org.elasticsearch.action.admin.cluster.state.
ClusterStateResponse]]; nested:
IndexOutOfBoundsException[Readable byte limit exceeded: 107];
```

- Check Elasticsearch Mappings

  You can influence the mapping by including a mapping-spec in the original CREATE command, but you cannot provide your own mapping. Be aware that this default mapping from Elasticsearch includes assumptions about data types and data structures in your documents. On the basis of these assumptions, Elasticsearch may omit your document from the index. For example, string mapped to object will not be indexed in the current version. Also, a record that contains only empty strings or nulls in the indexed fields will be omitted. For more information about expected data structures see: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

- The population of a new text index begins immediately when it is created. If the existing database is large, this operation can take quite a long time. Furthermore, when a text index is created, if other text indexes that were created earlier already exist, they too will be populated over again from scratch. This is because all text indexes share the single gateway in server to stream data to the ES cluster, and due to the newly added index, the gateway has to start from the very beginning, resulting re-populating all indexes. For these reasons it is best to create all the text indexes that you need at the same time, and preferably before the database has been populated.