Advanced topics in NoSql databases

S2/3 – Cassandra

Submitted by

Camara Abdoul-karim, Maxime de la Tour, Yuning LI

DIA1 31 Jan. 2021

Basic information

Code link: https://github.com/Yuning-LI/Nosql-Cassandra
Dataset Name : ENRON
Complexity: 1 (requires 5 simple queries, 2 complex queries, 2 hard queries)
Schema :

[datasets.enron]	100.0%	Collection
_id	100.0%	ObjectId
bcc	100.0%	Array
cc	100.0%	Array
ctype	100.0%	String
date	100.0%	String
fname	100.0%	String
folder	100.0%	String
fpath	100.0%	String
mid	100.0%	String
recipients	100.0%	Array
replyto	100.0%	Null
sender	100.0%	String
subject	100.0%	String
text	100.0%	String
to	100.0%	Array

Data cleaning

In order to import the dataset, we need to firstly organize the data format.
What we need to do is as follow:
1. add the CQL command 'INSERT INTO <tablename> JSON' to the beginning.
2. add '' and ; to each line
3. change the key "to" to "toaddress" as it can not be recognized by Cassandra
4. rewrite the key "_id" to simplify it.
5. rewrite the key "fname" and change it to integer type.

So we create a python script (dataCleaning.py) for data cleaning, the python code is follow:

dataCleaning.py | ctype_folder.py | index.js config | index.js fbeamer

Users > liyuning > Desktop > nosql > S2_Cassandra > dataCleaning.py > dataCleaning

```python
import json
def dataCleaning (input_json_file, output_json_file):
    file_in = open(input_json_file, "r")
    file_out = open(output_json_file, "w")
    data = file_in.readlines() # read each json line in the dataset

    for line in data:
        json_data = json.loads(line) # get the json object
        json_data['id'] = json_data['_id']['$oid'] # rewrite the key "_id"
        del json_data['_id']

        json_data['toaddress'] = json_data['to'] # rewrite the key "to"
        del json_data['to']

        json_data['fname'] = int(json_data['fname'][:-1]) # rewrite the key "fname"

        oldline = json.dumps(json_data) # transform the json object to string

        newline = 'INSERT INTO emails JSON \'' + oldline + '\';' + '\n' # add CQL command
        file_out.writelines(newline)

    file_in.close()
    file_out.close()

dataCleaning('/Users/liyuning/Desktop/nosql/enron.json', '/Users/liyuning/Desktop/nosql/enronClean.json')
```

After dealt by python, we get the dataset with the form as follow:


```
{
  "toaddress": ["sherri.reinartz@enron.com"],
  "sender": "rosalee.fleming@enron.com",
  "recipients": ["sherri.reinartz@enron.com"],
  "cc": [],
  "text": "Ken will attend both meetings.\n\nRosie\n\n\n\nSherri Reinartz\n01/12,
  "mid": "18133935.1075840283210.JavaMail.evans@thyme",
  "fpath": "enron_mail_20110402/maildir/lay-k/_sent/1.",
  "bcc": [],
  "replyto": null,
  "ctype": "text/plain; charset=us-ascii",
  "fname": 1,
  "date": "2000-01-12 08:24:00-08:00",
  "folder": "_sent",
  "id": "52af48b5d55148fa0c199643",
  "subject": "Re: EXECUTIVE COMMITTEE MEETINGS - MONDAY, JANUARY 17"
}
```


And the new json file “enronClean.json” as follow:

enronClean.json

```
INSERT INTO emails JSON '{"toaddress": ["sherri.reinartz@enron.com"], "sender": "rosalee.fleming@enron.com",
INSERT INTO emails JSON '{"toaddress": ["lizard_ar@yahoo.com"], "sender": "rosalee.fleming@enron.com", "reci
INSERT INTO emails JSON '{"toaddress": ["rob.bradley@enron.com"], "sender": "rosalee.fleming@enron.com", "re
INSERT INTO emails JSON '{"toaddress": ["michael@optsevents.com"], "sender": "tori.wells@enron.com", "recipi
INSERT INTO emails JSON '{"toaddress": ["jaime.alatorre@enron.com"], "sender": "rosalee.fleming@enron.com",
INSERT INTO emails JSON '{"toaddress": ["loyal_lay@yahoo.com"], "sender": "rosalee.fleming@enron.com", "reci
INSERT INTO emails JSON '{"toaddress": ["michael.mann@enron.com"], "sender": "rosalee.fleming@enron.com", "r
INSERT INTO emails JSON '{"toaddress": ["michael.mann@enron.com"], "sender": "rosalee.fleming@enron.com", "r
INSERT INTO emails JSON '{"toaddress": ["tori.wells@enron.com"], "sender": "tori.wells@enron.com", "recipien
INSERT INTO emails JSON '{"toaddress": ["lizard_ar@yahoo.com"], "sender": "rosalee.fleming@enron.com", "reci
INSERT INTO emails JSON '{"toaddress": ["ted.enloe@compaq.com"], "sender": "rosalee.fleming@enron.com", "rec
INSERT INTO emails JSON '{"toaddress": ["lizard_ar@yahoo.com"], "sender": "rosalee.fleming@enron.com", "reci
INSERT INTO emails JSON '{"toaddress": ["katherine.brown@enron.com"], "sender": "rosalee.fleming@enron.com",
INSERT INTO emails JSON '{"toaddress": ["mmfoss@uh.edu"], "sender": "rosalee.fleming@enron.com", "recipients
INSERT INTO emails JSON '{"toaddress": ["michael.hicks@enron.com"], "sender": "rosalee.fleming@enron.com", "
INSERT INTO emails JSON '{"toaddress": ["chmoore1@email.msn.com"], "sender": "rosalee.fleming@enron.com", "r
INSERT INTO emails JSON '{"toaddress": ["diane.bazelides@enron.com"], "sender": "rosalee.fleming@enron.com",
INSERT INTO emails JSON '{"toaddress": ["amy.lee@enron.com"], "sender": "rosalee.fleming@enron.com", "recipi
INSERT INTO emails JSON '{"toaddress": ["dyergin@cera.com"], "sender": "rosalee.fleming@enron.com", "recipie
INSERT INTO emails JSON '{"toaddress": ["tori.wells@enron.com", "vanessa.groscrand@enron.com", "sally.keeper
INSERT INTO emails JSON '{"toaddress": ["darlenet23@earthlink.net"], "sender": "rosalee.fleming@enron.com",
INSERT INTO emails JSON '{"toaddress": ["chmoore1@email.msn.com"], "sender": "rosalee.fleming@enron.com", "r
INSERT INTO emails JSON '{"toaddress": ["joe.hillings@enron.com"], "sender": "rosalee.fleming@enron.com", "r
```

Then we create the table "emails"

```
CREATE TABLE emails (
    id TEXT,
    sender TEXT,
    recipients LIST<TEXT>,
    cc LIST<TEXT>,
    text TEXT,
    mid TEXT,
    fpath TEXT,
    bcc LIST<TEXT>,
    toaddress LIST<TEXT>,
    replyto TEXT,
    ctype TEXT,
    fname INT,
    date TEXT,
    folder TEXT,
    subject TEXT,
    PRIMARY KEY (sender)
);

ALTER TABLE emails WITH GC_GRACE_SECONDS = 0;
```

Import the json file from CLI

Items | Queries | History

Search for item...

emails

id	bcc	cc	ctype
52af48b6d55148fa0c199aec	{"dodfraser@aol.com"}	{"dodfraser@aol.com"}	text/plain; charset=us-a
52af48b6d55148fa0c199b7b	{"connie.levo@compaq.com"}	{"connie.levo@compaq.com"}	text/plain; charset=us-a
52af48b6d55148fa0c1998f1	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c199734	{"rosalee.fleming@enron.com"}	{"rosalee.fleming@enron.com"}	text/plain; charset=us-a
52af48b6d55148fa0c199a19	{"brown_mary_jo@lilly.com"}	{"brown_mary_jo@lilly.com"}	text/plain; charset=us-a
52af48b6d55148fa0c199928	{"david.forster@enron.com", "sheri.thomas@enron....	{"david.forster@enron.com", "sheri.thomas@enron....	text/plain; charset=ANSI
52af48b6d55148fa0c199ab5	{"kenneth.lay@enron.com"}	{"kenneth.lay@enron.com"}	text/plain; charset=us-a
52af48b6d55148fa0c1996e3	{"jeff.skilling@enron.com", "andrew.fastow@enron.c...	{"jeff.skilling@enron.com", "andrew.fastow@enron.c...	text/plain; charset=us-a
52af48b6d55148fa0c199ad3	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c199af6	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c199b3b	NULL	NULL	text/plain; charset=ANSI
52af48b6d55148fa0c199a85	{"brown_mary_jo@lilly.com", "decoudreaux_alecia_a...	{"brown_mary_jo@lilly.com", "decoudreaux_alecia_a...	text/plain; charset=us-a
52af48b6d55148fa0c19976f	{"mark.palmer@enron.com"}	{"mark.palmer@enron.com"}	text/plain; charset=us-a
52af48b6d55148fa0c199779	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c19981d	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c199918	{"tori.wells@enron.com"}	{"tori.wells@enron.com"}	text/plain; charset=us-a
52af48b6d55148fa0c1998f2	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c19996f	{"dorothy.barnes@enron.com"}	{"dorothy.barnes@enron.com"}	text/plain; charset=us-a
52af48b6d55148fa0c1996c5	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c1998b1	{"james.derrick@enron.com", "kelly.communications...	{"james.derrick@enron.com", "kelly.communications...	text/plain; charset=ANSI
52af48b6d55148fa0c199b93	{"jdavidson@srfunds.com", "rosalee.fleming@enron...	{"jdavidson@srfunds.com", "rosalee.fleming@enron...	text/plain; charset=us-a
52af48b6d55148fa0c199b32	NULL	NULL	text/plain; charset=us-a
52af48b6d55148fa0c199a2b	NULL	NULL	text/plain; charset=us-a

Simple queries * 5

1. List of the senders' email addresses.

```
24 SELECT sender FROM emails;
25
26
```

line 24, column 27, loca… Beautify ⌘I Run Current ⌘↵

sender
shea_dugger@i2.com
marilyn.chalmers@compaq.com
rosalee.fleming@enron.com
mark.lay@enron.com
michael.mann@enron.com
rosalee.fleming@enron.com
doug.leach@enron.com

2. Number of the emails sent from "tom.siekman@compaq.com"

```
26 SELECT COUNT(*) FROM emails WHERE sender='tom.siekman@compaq.com' ALLOW
   FILTERING;
27
```

line 26, column 82, loca… Beautify ⌘I Run Current ⌘↵

count
5

3. Number of the emails where folder equals to 'inbox' and the recipients contains 'kenneth.lay@enron.com'

```
29 SELECT COUNT(*) FROM emails WHERE folder='inbox' AND recipients CONTAINS
   'kenneth.lay@enron.com' ALLOW FILTERING;
30
```

line 29, column 113, lo… Beautify ⌘I Run Current ⌘↵

count
203

4. The content of the email sent from jeffrey.westphal@enron.com at the date 2001-09-26

```
30 SELECT text FROM emails WHERE sender='jeffrey.westphal@enron.com' AND date='2001-09-26 09:02:02-07:00'
   ALLOW FILTERING;
31
```

line 30, column 93, location 837 Beautify ⌘I Run Current ⌘↵

text
Well: You know.... No one expected the World Trade Centre to be destroyed by terrorists either...I think making a statement like "We do not ex.

5. See on which days did the address 'kenneth.lay@enron.com' receive new emails.

```
31 SELECT date FROM emails WHERE recipients CONTAINS 'kenneth.lay@enron.com'
    ALLOW FILTERING;
32
```

line 31, column 91, locati… Beautify ⌘I Run Current ⌘↵

date
2001-11-15 13:43:02-08:00
2000-08-21 02:36:00-07:00
2001-09-26 09:02:02-07:00
2001-05-25 13:07:11-07:00
2001-12-03 11:27:58-08:00
2000-10-11 05:22:00-07:00

Complex queries * 2

1. Count the number of the emails per address

```
29 SELECT sender, COUNT(*) FROM emails GROUP BY sender;
30
31
```

line 19, column 21, loca… Beautify ⌘I Run Current ⌘↵

sender	count
ricex@swbell.net	1
jeff.donahue@enron.com	1
corry.bentley@enron.com	1
jeffrey.westphal@enron.com	1
esheridan@uh.edu	1
40enron@enron.com	1

29 ms 504 rows Message Export...

2. Distribution of text types for the emails in the folder "notes_inbox"

In order to avoid the single primary key problem, we create another table containing "ctype" and "folder" information.

```
32 DROP TABLE ctype_folder;
33 CREATE TABLE ctype_folder (
34     id TEXT,
35     ctype TEXT,
36     folder TEXT,
37     PRIMARY KEY ((ctype), folder)
38 );
39
40 ALTER TABLE emails WITH GC_GRACE_SECONDS = 0;
41 CREATE INDEX btree_ctype_folder_id on ctype_folder(id);
```

Create another python file (ctype_folder.py) to reform the original data

dataCleaning.py | ctype_folder.py | index.js config | index.js fbeamer

Users > liyuning > Desktop > nosql > S2_Cassandra > ctype_folder.py > dataReform

```python
import json
def dataReform (input_json_file, output_json_file):
    file_in = open(input_json_file, "r")
    file_out = open(output_json_file, "w")
    data = file_in.readlines() # read each json line in the dataset

    for line in data:
        json_data = json.loads(line) # get the json object
        newjson = {}
        newjson['id'] = json_data['_id']['$oid']
        newjson['ctype'] = json_data['ctype']
        newjson['folder'] = json_data['folder']

        oldline = json.dumps(newjson) # transform the json object to string

        # add the CQL insert command
        newline = 'INSERT INTO ctype_folder JSON \'' + oldline + '\';' + '\n'
        file_out.writelines(newline)

    file_in.close()
    file_out.close()

dataReform('/Users/liyuning/Desktop/nosql/enron.json', '/Users/liyuning/Deskto
```

Import the json file at CLI and insert the table

dblp

enron

Items | Queries | History

Search for item...

ctype_folder

emails

SQL Query

ctype_folder

ctype	folder	id
text/plain; charset=ANSI_X3.4-1968	_sent	52af48b6d55148fa0c199731
text/plain; charset=ANSI_X3.4-1968	all_documents	52af48b6d55148fa0c199b98
text/plain; charset=ANSI_X3.4-1968	business	52af48b6d55148fa0c199bab
text/plain; charset=ANSI_X3.4-1968	deleted_items	52af48b6d55148fa0c199ee6
text/plain; charset=ANSI_X3.4-1968	discussion_threads	52af48b6d55148fa0c19a396
text/plain; charset=ANSI_X3.4-1968	elizabeth	52af48b6d55148fa0c19a3c2
text/plain; charset=ANSI_X3.4-1968	enron	52af48b6d55148fa0c19a3cb
text/plain; charset=ANSI_X3.4-1968	family	52af48b6d55148fa0c19a3d4
text/plain; charset=ANSI_X3.4-1968	inbox	52af48b7d55148fa0c19a920
text/plain; charset=ANSI_X3.4-1968	notes_inbox	52af48b7d55148fa0c19ac45
text/plain; charset=ANSI_X3.4-1968	sent	52af48b7d55148fa0c19ad5c
text/plain; charset=ANSI_X3.4-1968	sent_items	52af48b7d55148fa0c19ad60
text/plain; charset=us-ascii	_sent	52af48b6d55148fa0c199747
text/plain; charset=us-ascii	all_documents	52af48b6d55148fa0c199ba9
text/plain; charset=us-ascii	business	52af48b6d55148fa0c199baa
text/plain; charset=us-ascii	calendar	52af48b6d55148fa0c199bb3
text/plain; charset=us-ascii	compaq	52af48b6d55148fa0c199bb4
text/plain; charset=us-ascii	deleted_items	52af48b6d55148fa0c19a01a
text/plain; charset=us-ascii	discussion_threads	52af48b6d55148fa0c19a3a5
text/plain; charset=us-ascii	elizabeth	52af48b6d55148fa0c19a3c8
text/plain; charset=us-ascii	enron	52af48b6d55148fa0c19a3cd
text/plain; charset=us-ascii	family	52af48b6d55148fa0c19a3d5

Imply the query

```
32 DROP TABLE ctype_folder;
33 CREATE TABLE ctype_folder (
34     id TEXT,
35     ctype TEXT,
36     folder TEXT,
37     PRIMARY KEY ((ctype), folder)
38 );
39
40 ALTER TABLE emails WITH GC_GRACE_SECONDS = 0;
41 CREATE INDEX btree_ctype_folder_id on ctype_folder(id);
42 SELECT ctype FROM ctype_folder WHERE folder='notes_inbox' GROUP BY ctype
    ALLOW FILTERING;
43
44
```

line 43, column 1, loca… Beautify ⌘I Run Current ⌘↵

ctype
text/plain; charset=ANSI_X3.4-1968
text/plain; charset=us-ascii

Hard queries * 2

1. the first recipient of the email with subject "Anonymous report on violations by senior Enron officials"

Modify the Cassandra.yaml config file to allow user define function:

liyuning — root@f462f27bde12: /etc/cassandra — com.docker.cli ◂ docker exe...

```
# This threshold can be adjusted to minimize logging if necessary
# gc_log_threshold_in_ms: 200

# If unset, all GC Pauses greater than gc_log_threshold_in_ms will log at
# INFO level
# UDFs (user defined functions) are disabled by default.
# As of Cassandra 3.0 there is a sandbox in place that should prevent execution
of evil code.
enable_user_defined_functions: true

# Enables scripted UDFs (JavaScript UDFs).
# Java UDFs are always enabled, if enable_user_defined_functions is true.
# Enable this option to be able to use UDFs with "language javascript" or any cu
stom JSR-223 provider.
# This option has no effect, if enable_user_defined_functions is false.
enable_scripted_user_defined_functions: false

# The default Windows kernel timer and scheduling resolution is 15.6ms for power
 conservation.
# Lowering this value on Windows can provide much tighter latency and better thr
oughput, however
# some virtualized environments may see a negative performance impact from chang
ing this setting
"cassandra.yaml" 1279L, 59843C                                1110,1        87%
```

Restart Cassandra server, define function to return the first element of the list, imply the query:

```
33 CREATE OR REPLACE FUNCTION projectList (num INT, tab LIST<TEXT>)
34     RETURNS NULL ON NULL INPUT RETURNS TEXT LANGUAGE Java
35     AS 'return (String)tab.get(num);';
36 SELECT projectList(0, toaddress) AS first_recipient FROM emails WHERE
   subject='Anonymous report on violations by senior Enron officials' ALLOW
   FILTERING;
```

line 36, column 52, location 1… Beautify ⌘I Run Current ⌘↵

first_recipient
jskilli@enron.com

2. Compute the average fname of the emails sent to 'kenneth.lay@enron.com'

Define User Define Aggregate functions

```
40 CREATE OR REPLACE FUNCTION avgState ( state tuple<int,bigint>, val int )
   CALLED ON NULL INPUT RETURNS tuple<int,bigint> LANGUAGE java
41 AS 'if (val !=null) {
42         state.setInt(0, state.getInt(0)+1);
43         state.setLong(1, state.getLong(1)+val.intValue()); }
44     return state;';
45
46 CREATE OR REPLACE FUNCTION avgFinal ( state tuple<int,bigint> ) CALLED ON NULL
   INPUT RETURNS double LANGUAGE java
47 AS 'double r = 0;
48     if (state.getInt(0) == 0) return null;
49     r = state.getLong(1);
50     r/= state.getInt(0);
51     return Double.valueOf(r);';
52
53 CREATE AGGREGATE IF NOT EXISTS average ( int )
54     SFUNC avgState STYPE tuple<int,bigint>
55     FINALFUNC avgFinal INITCOND (0,0);
56
57 SELECT average(fname) FROM emails WHERE toaddress CONTAINS
   'kenneth.lay@enron.com' ALLOW FILTERING;
58
```

line 57, column 99, location 2… Beautify ⌘I Run Current ⌘↵

enron.average(fname)
688.8919860627178