

# [S25] IBD A2 Report

Matevosian Artem, B22-DS-01

## Methodology: Data preparation

The prepare\_data.py from the original assignment statement managed to satisfy my needs after uncommenting writing to csv operation, therefore, I directly used the script from the assignment - it reads parquet file, selects 1000 entries, creates docs + writes to csv in the format <doc\_id>\t<title>\t<text>

- I spent an abnormal amount of time trying to fix Out-Of-Memory errors that appeared independently of spark vectorized reader configs, later these reduced in frequency when I started deleting all the files from the previous runs right before main script execution.

## Methodology: Indexer

The result after running indexing: cassandra-server gets the tables as listed below (P = partitioning key, C = clustering key)

term_frequencies		
term	TEXT	P
document_id	INT	C (ASC)
tf	INT	

document_frequencies		
term	TEXT	P
df	INT	

doc_index		
document_id	INT	P
title	TEXT	
length	INT	

Why exactly these tables?

- term\_frequencies = **tf(t, d)**
- document\_frequencies = **df(t)**
- doc\_index serves as mapping of document\_id to title AND as **dl(d)**

For each table, there is a specific mapreduce job definition, ordered as below:

1. term\_frequencies
2. document\_frequencies
3. doc\_index

The mapreduce jobs print out to /index/data in HDFS in the format directly transferable to cassandra, given the table definitions as above. There were multiple options to copy these outputs into Cassandra:

- (Selected by me) run consecutive INSERTS for every entry – able to run from wherever we have access to cassandra-server, no need to load the whole tsv locally when reading it lazily from HDFS
- run CQLSH COPY on TSVs (rejected since it supposedly requires to load outputs from HDFS into local filesystem)
- sstableloader after building same tables on the running machine (rejected for this very reason, cassandra-server should store the tables, I did not want to create them “locally” and then push somewhere else)

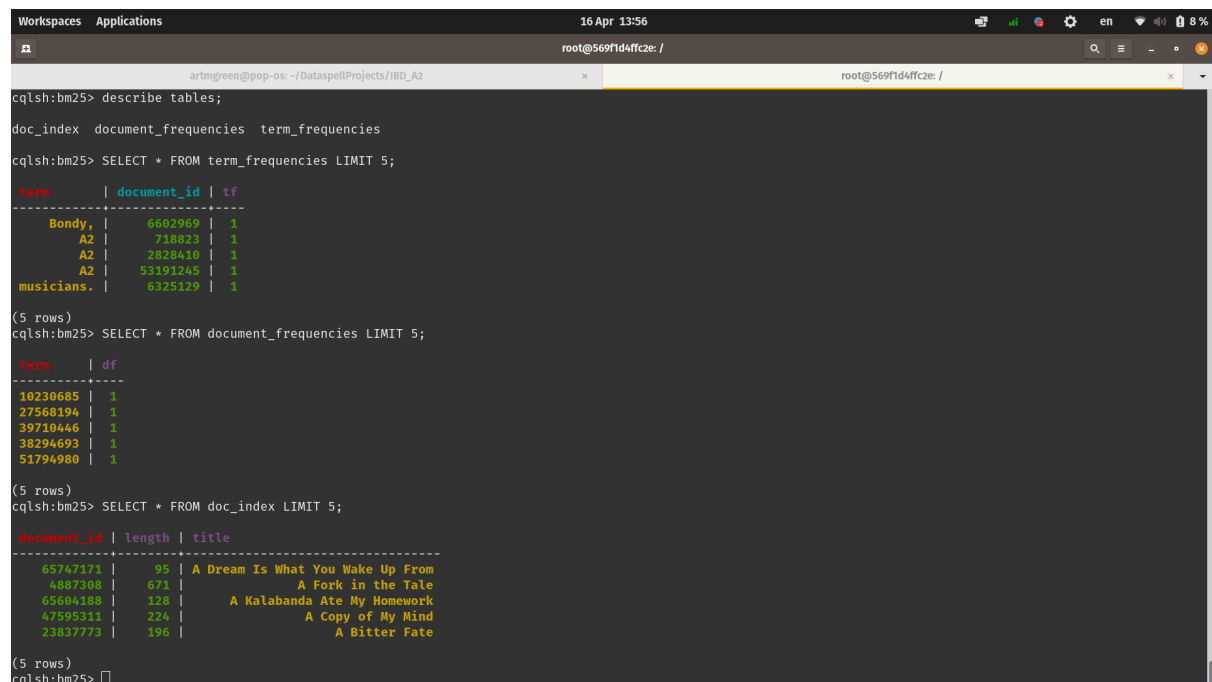
As a result, there are three tables defined as above.

NOTE: document\_frequencies could be created much more quickly via  
SELECT term, COUNT(document\_id) FROM term\_frequencies GROUP BY term;

However, according to Apache’s docs, Cassandra’s materialized views prohibit aggregate functions in their definitions. Also, printing the query result into a TSV for a subsequent CQLSH COPY may or may not be faster than running the second MapReduce pipeline, so I stayed with the second pipeline. Finally, if SQL-based DBs had been available for the assignment (e.g. Postgres), it would have been possible to CREATE table AS SELECT statement as shown above.

## Demonstration: indexer’s result

For the purposes of demonstration of indexer’s results, this is the screenshot regarding Cassandra’s tables (I executed bash in cassandra-server container to be able to CQLSH it):



```
Workspaces Applications 16 Apr 13:56
root@569f1d4ffc2e: /

artmgreen@pop-os: ~/DataspellProjects/IBD_A2
root@569f1d4ffc2e: /

cqlsh:bm25> describe tables;

doc_index  document_frequencies  term_frequencies

cqlsh:bm25> SELECT * FROM term_frequencies LIMIT 5;

term      | document_id | tf
-----|-----|-----
Bondy,    | 6602969    | 1
A2        | 718823     | 1
A2        | 2828410    | 1
A2        | 53191245   | 1
musicians.| 6325129    | 1

(5 rows)
cqlsh:bm25> SELECT * FROM document_frequencies LIMIT 5;

term      | df
-----|-----
10230685  | 1
27568194  | 1
39710446  | 1
38294693  | 1
51794980  | 1

(5 rows)
cqlsh:bm25> SELECT * FROM doc_index LIMIT 5;

document_id | length | title
-----|-----|-----
65747171    | 95     | A Dream Is What You Wake Up From
4887308     | 671    | A Fork in the Tale
65604188    | 128    | A Kalabanda Ate My Homework
47595311    | 224    | A Copy of My Mind
23837773    | 196    | A Bitter Fate

(5 rows)
cqlsh:bm25>
```

# Methodology: Ranker

NOTE: the methodology below is how I was trying to implement the ranker. In practice, it runs, but returns zero scores due to an error I was not able to figure out yet.

Previous approach: wrap everything into a BM25\_Calculator class and broadcast it over the workers. Rejected since the class contains Cassandra session pointer, which cannot be packed (“pickled”) to broadcast.

Current approach that one can see in query.py:

init\_cassandra(): a function to give session credentials to whoever called it, e.g. workers to receive their own session (since it cannot be serialized and passed from main executor).

Additionally it computes  $N$  and  $dl_{avg}$  (see query.py)

compute\_bm25(query\_terms, doc\_id):

1. receives its own Cassandra session
2. checks if the suggested document exists at all (and returns empty title + 0 score if it does not)
3. iterates over every unique term in query and computes “scary-looking formula” from the assignment statement (+0.5 in logarithm numerator and denominator to prevent numerical errors, given that logarithms do not act nicely with zeros) for every term, sums over the terms and returns the sum with doc info
4. Shutdowns session — I do not want to deal with errors that may or may not appear if it does not.

Top-level code:

1. parses query into terms
2. configures Spark
3. receives its own Cassandra session
4. finds all the documents that have non-zero chance to be relevant ( $tf(doc, term) \neq 0$ )
5. computes BM25 in parallel (query is broadcasted, Cassandra session is created by workers on their own. Broadcasting  $N$  and  $dl_{avg}$  is a leftover for the testing purposes)

NOTE: given that the ranker does not work properly (zero scores in every document), I cannot provide a proper demonstration. However, running app.sh via cluster-master should be able run both indexer and ranker.