BigData Assignment 2 report

April 2025

Assignment goal is to create a full-text search engine that can index a collection of documents in large volume and generate appropriate ranked responses for user searches. The steps are:

Data Preparation: Spark is employed to preprocess a massive Parquet dataset by document sampling and document conversion to plain text files.

Indexing: Hadoop MapReduce programs tokenized documents, calculate term frequencies/positions, compute document frequency, and extract document metadata.

Data Persistence: Processed index data is loaded into Apache Cassandra. I created three tables: term_document, documents_info and document_frequency.

1. table `term_document` - contains term frequency in particular document and positions in symbols for each term in the document:

```
CREATE TABLE IF NOT EXISTS term_document (
    term text,
    doc_id text,
    tf int,
    positions list<int>,
    PRIMARY KEY (term, doc_id)
)
```

2. table `documents_info` - has documents id, titles and length of words in document:

```
CREATE TABLE IF NOT EXISTS documents_info (
    doc_id text PRIMARY KEY,
    title text,
    length int
)
```

3. table `document_frequency` - the same as vocabulary of terms, also include number of documents which has particular term:

```
CREATE TABLE IF NOT EXISTS document_frequency (
    term text PRIMARY KEY,
    df int
)
```

Query Processing: Query Processing: Apache Spark is used to process user queries to aggregate BM25 scores.

Containerization: Containerization: Docker Compose establishes a multi-container environment for nodes of the Spark cluster and for the Cassandra server.

1 Methodology

1.1 Data Preparation

The preparation of data is carried out in prepare_data.py and prepare_data.sh. It loads a Parquet file using Apache Spark, from which a sample of documents is obtained.

Each document title is normalized by spaces, because some of them contained $\backslash n$ or $\backslash t$ symbols that will break the division in future steps (since we divide doc_id, title and text using $\backslash t$ symbol in csv file).

After that we specify the filenames in the format `<doc_id>_<title >.txt`. We put the text to the file and save it in data folder in hdfs.

Additionally, we save *.tsv* file that contain rows with documents in the following format `<doc_id \t title \t text>`

1.2 Indexing using MapReduce pipelines

I have 3 pairs of MapReduce files to perform the indexing:

Job 1: Term Frequencies and Positions

Mapper: (`mapper1.py`) Tokenized the document content and output each token along with its document ID and positional index.
Reducer: (`reducer1.py`) I grouped tokens by term and document ID, aggregated their positions, and output the term, document ID, term frequency, and a comma-separated list of positions.

Job 2: Document Frequencies

Mapper: (`mapper2.py`) Extract the term and document id from Job 1 output.
Reducer: (`reducer2.py`) Aggregates unique document ids per term and outputs the document frequency.

Job 3: Document Metadata

Mapper: (`mapper3.py`) Processes each document to calculate its length (total word or token count) and outputs the metadata (document id, title, length). To tokenize we convert text to lowercase, remove special characters, and split it by whitespaces.
Reducer: (`reducer3.py`) Passes the metadata output directly.

1.3 Data Import into Apache Cassandra

In `app.py` I established a connection to the Cassandra cluster, created the keyspace and three required tables, read the MapReduce output from HDFS using shell commands and executed batch insertions into Cassandra. If a batch is too large, it falls back to individual insertions.

1.4 Query Processing and BM25 Ranking

During the query phase (`query.py`), a user's search query is handled through the following steps:

1. The query is first normalized by converting it to lowercase and splitting it into tokens using regular expressions.

2. For each token, the system retrieves the document frequency and term frequencies from Cassandra.

3. The BM25 relevance score is calculated using the formula:

$$\log\left(\max\left(1, \frac{\text{doc\_count}}{\max(1, \text{df})}\right)\right)$$

$$score = idf \times \frac{tf \times (k1 + 1)}{tf + k1 \times \left(1 - b + b \times \frac{doc\_length}{avg\_doc\_length}\right)}$$

where `idf` is the inverse document frequency, `tf` is the term frequency, and `doc_length` and `avg_doc_length` are used to adjust for document length. `k1 = 1.0` and `b = 0.75`

4. Spark RDDs are used to aggregate BM25 scores for each document.
5. Finally, the documents with the highest BM25 scores are returned, showing the document ID, title, and score for each.

1.5 Containerization with Docker Compose

The `docker-compose.yml` file orchestrates the multi-container setup:

- **Cluster Master**: Runs the Spark master node and executes the application orchestration script (`app.sh`).
- **Cluster Slaves**: Two Spark slave nodes enhance the distributed processing capabilities.
- **Cassandra Server**: Provides the distributed storage layer for the index.

All containers are connected over a dedicated network allowing seamless communication between Spark and Cassandra.

2 Demonstration

2.1 Running the System

To run the system, follow these steps:

1. Clone the repository and navigate to the root directory.
2. Ensure you have Docker and Docker Compose installed and at least 10 GB of RAM on your machine.
3. Place the Parquet file (e.g., `a.parquet`) in the `./app` folder. Also change the name of the parquet file in the following line of `app.sh`: `bash prepare_data.sh a.parquet` if you use another file
4. Start the containers by running: `docker-compose up`
5. If you want to run paricular query, change it in `app.sh` in the following line: `bash search.sh "Your query"`

2.2 Screenshots

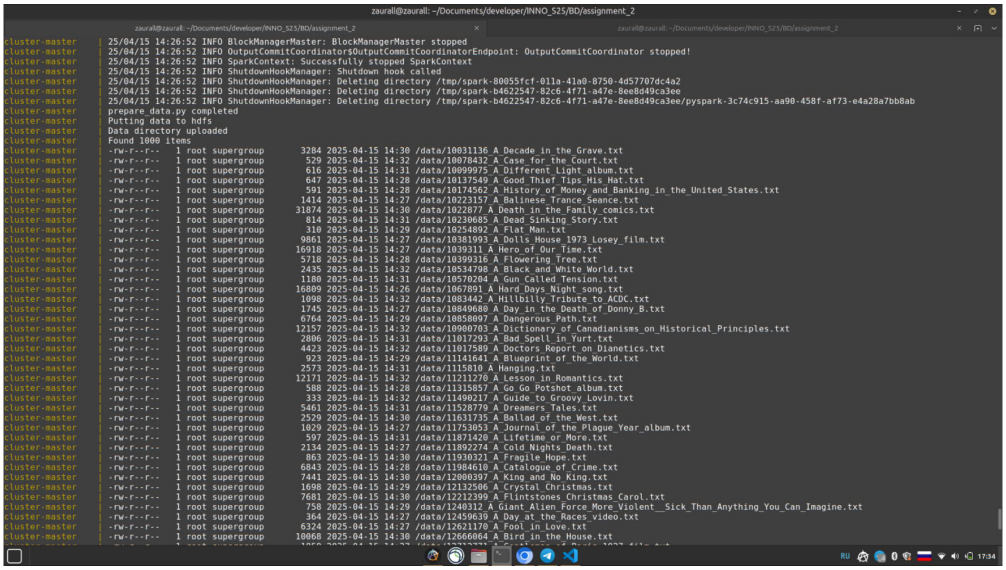

Figure 1: HDFS ls /data.

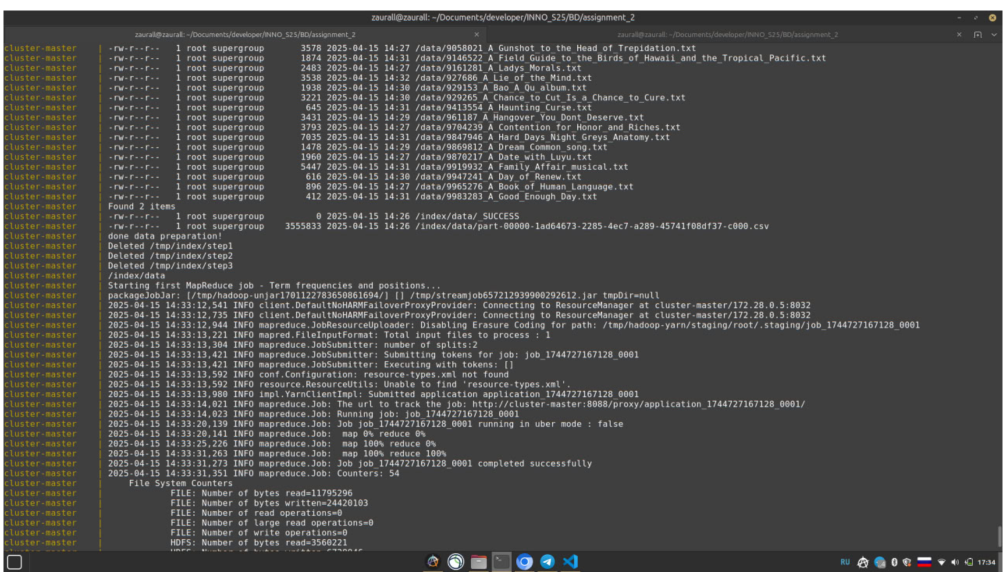

Figure 2: HDFS ls /index/data.

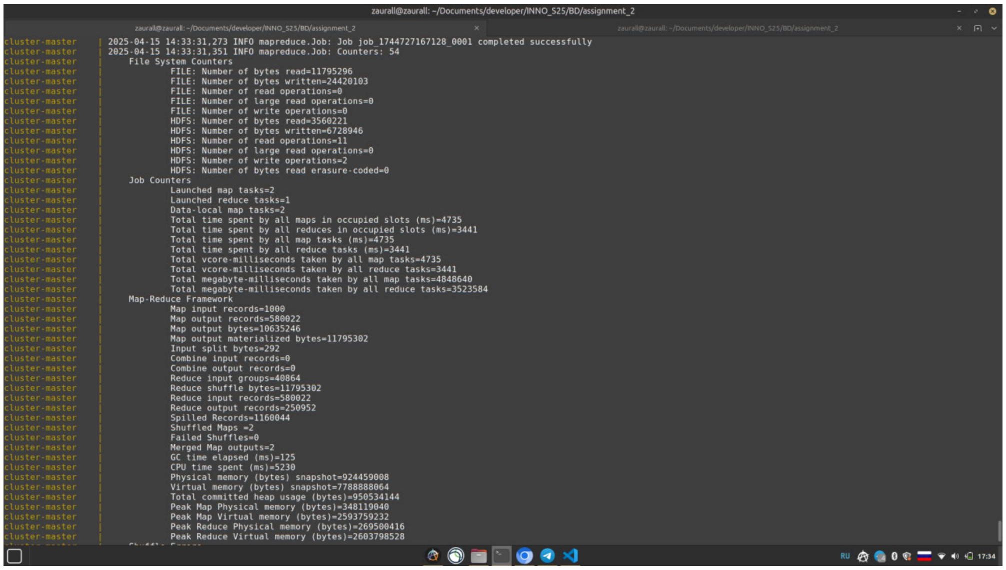

Figure 3: MapReduce pipeline 1

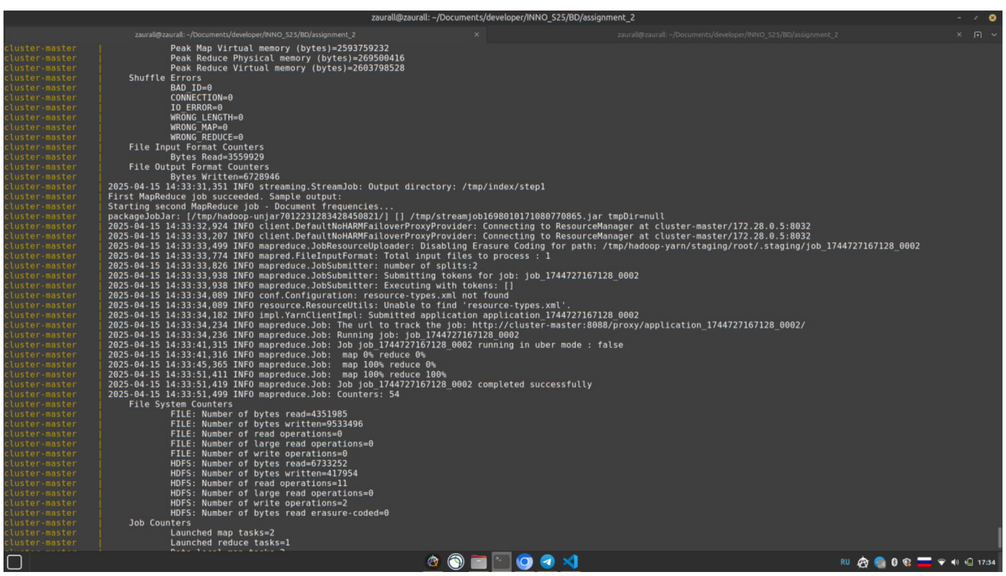

Figure 4: MapReduce pipeline 2

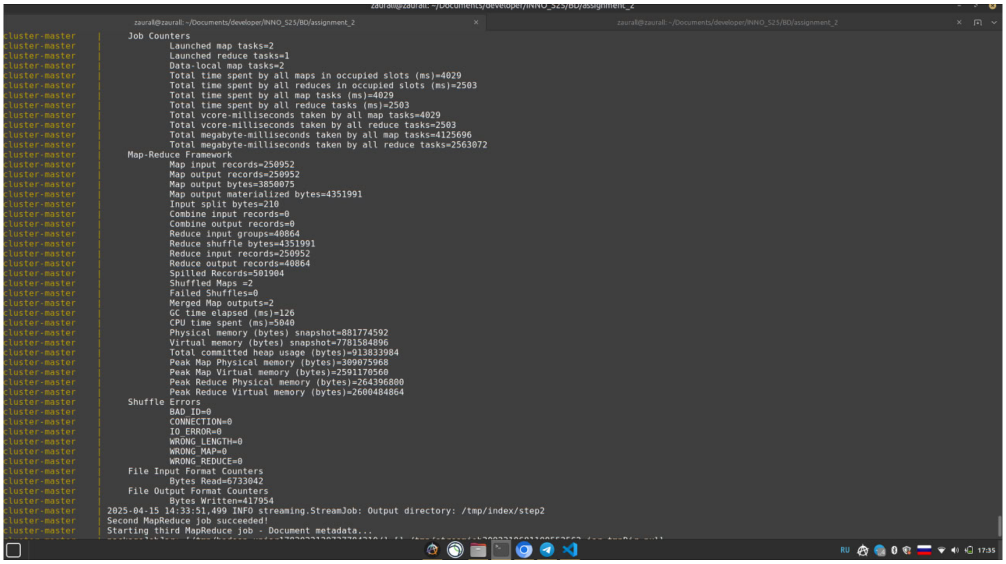

Figure 5: MapReduce pipeline 2

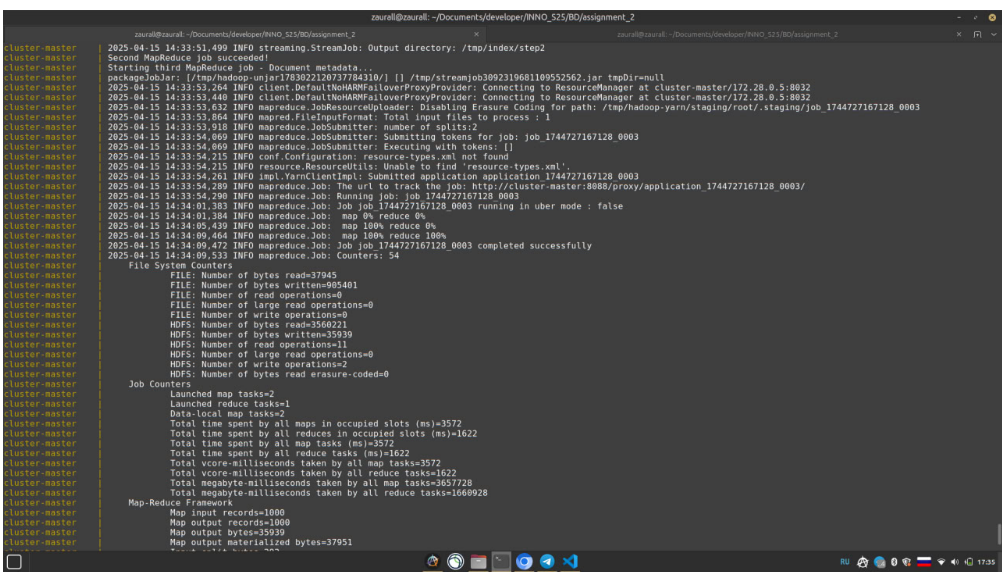

Figure 6: MapReduce pipeline 3

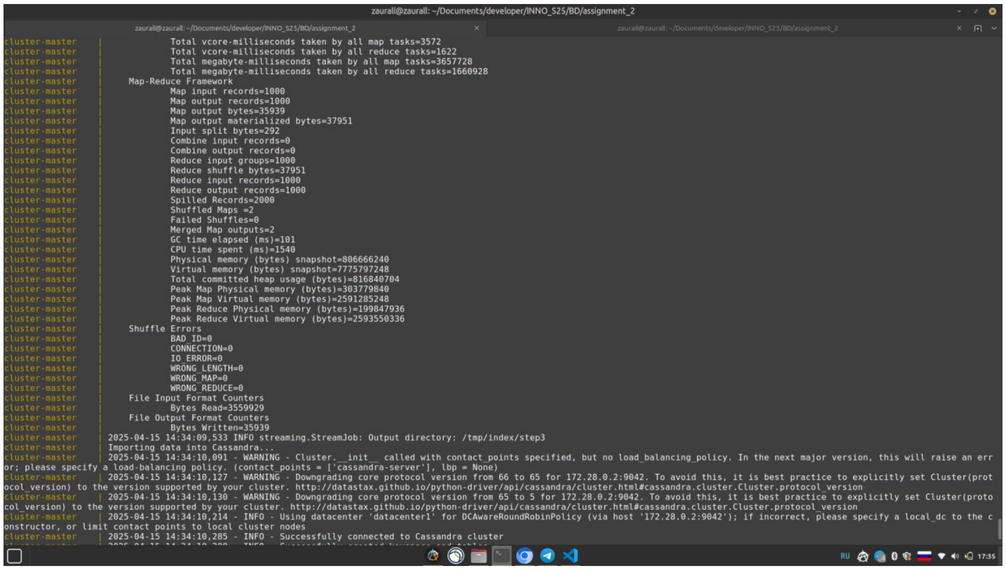

Figure 7: MapReduce pipeline 3

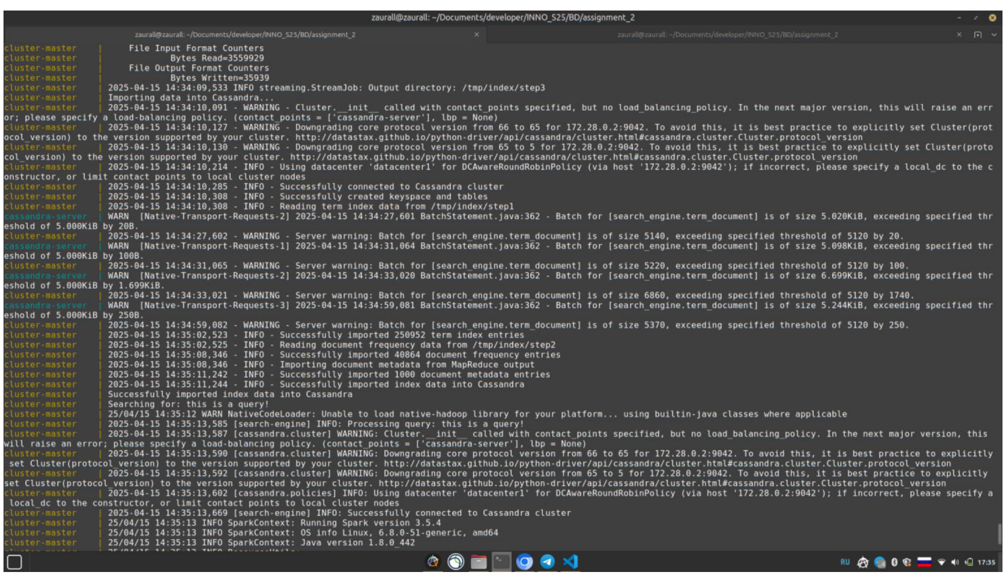

Figure 8: import index data to Cassandra

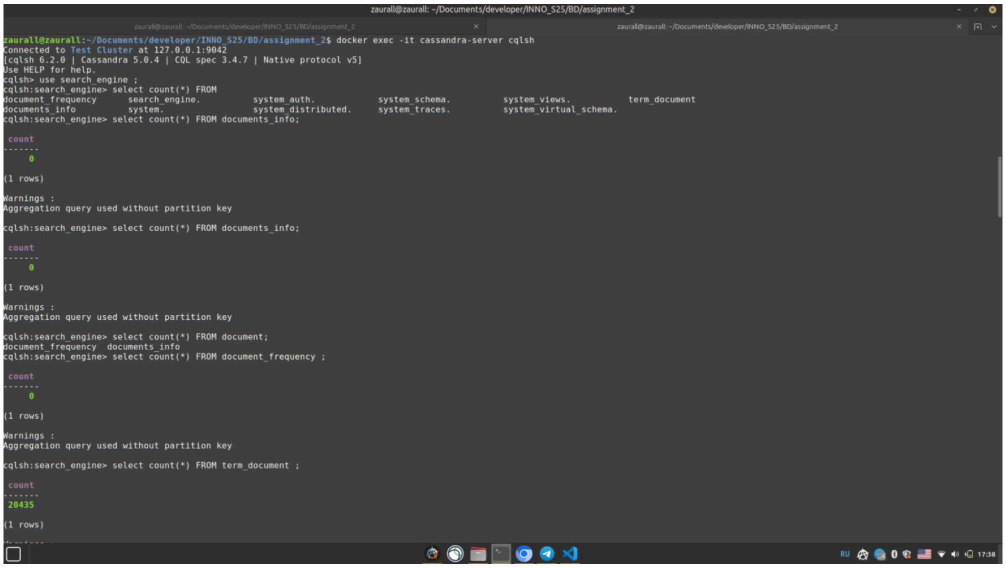

Figure 9: Cassandra tables content

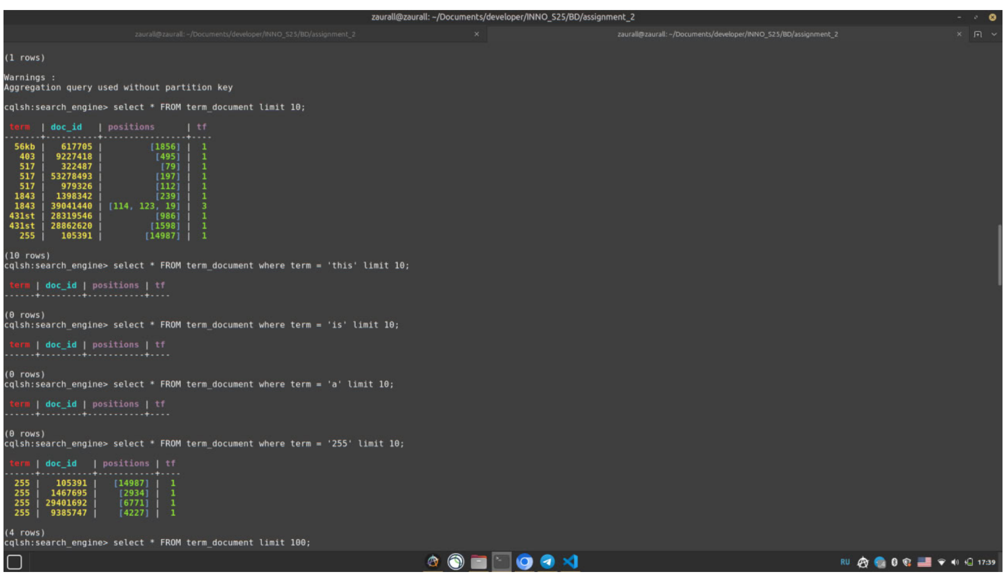

Figure 10: Cassandra tables content

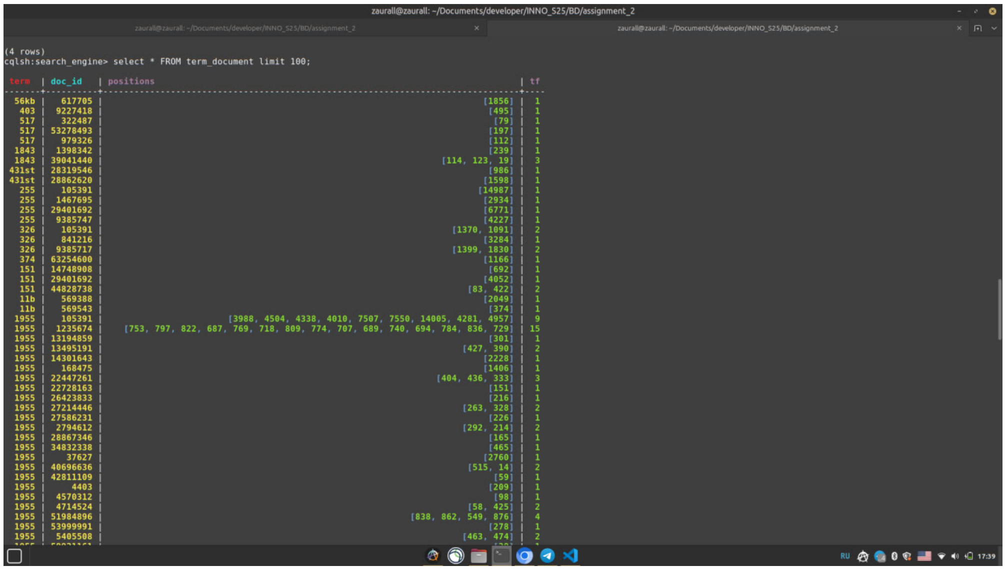

Figure 11: Cassandra tables content

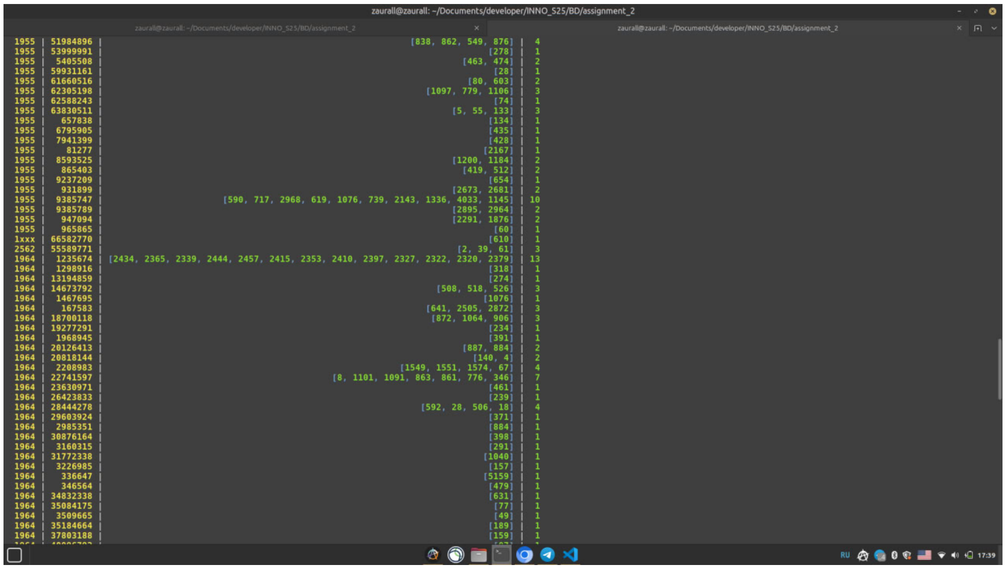

Figure 12: Cassandra tables content

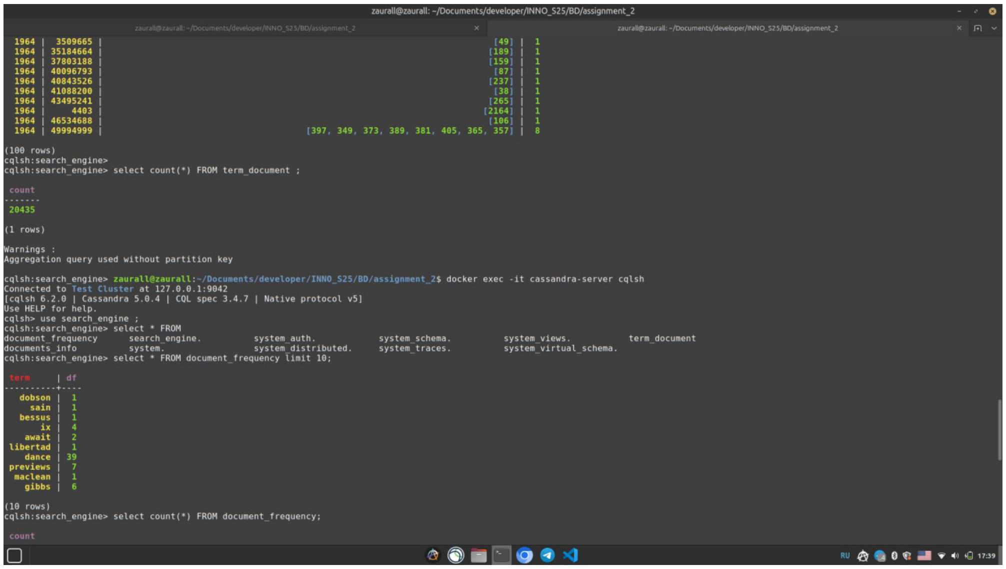

Figure 13: Cassandra tables content

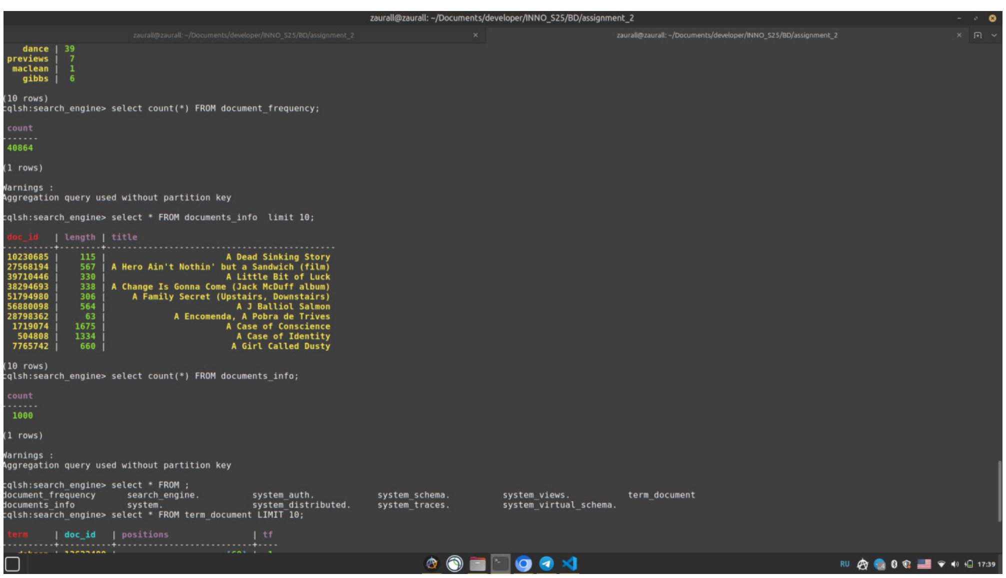

Figure 14: Cassandra tables content

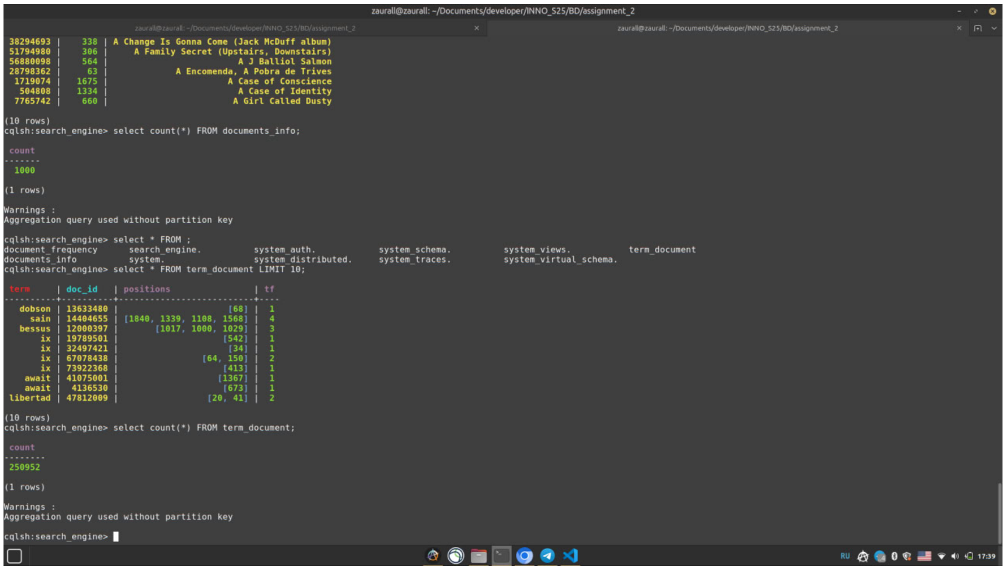

Figure 15: Cassandra tables content

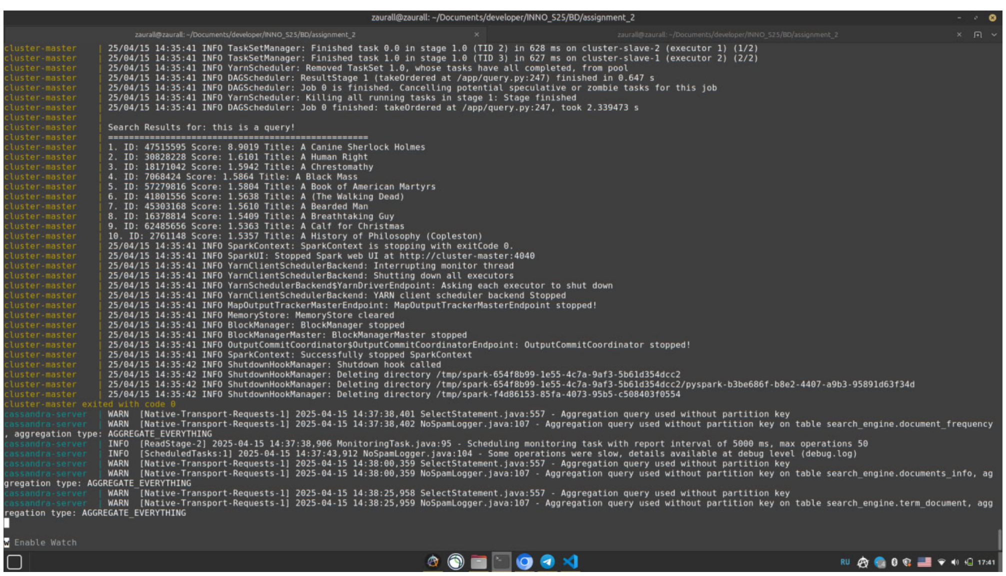

Figure 16: Query 1 results

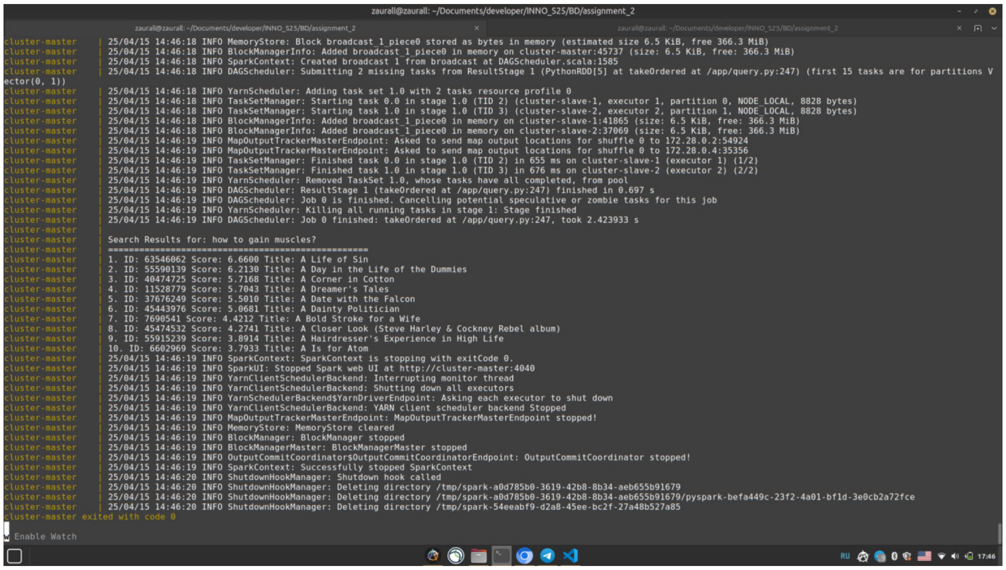

Figure 17: Query 2 results

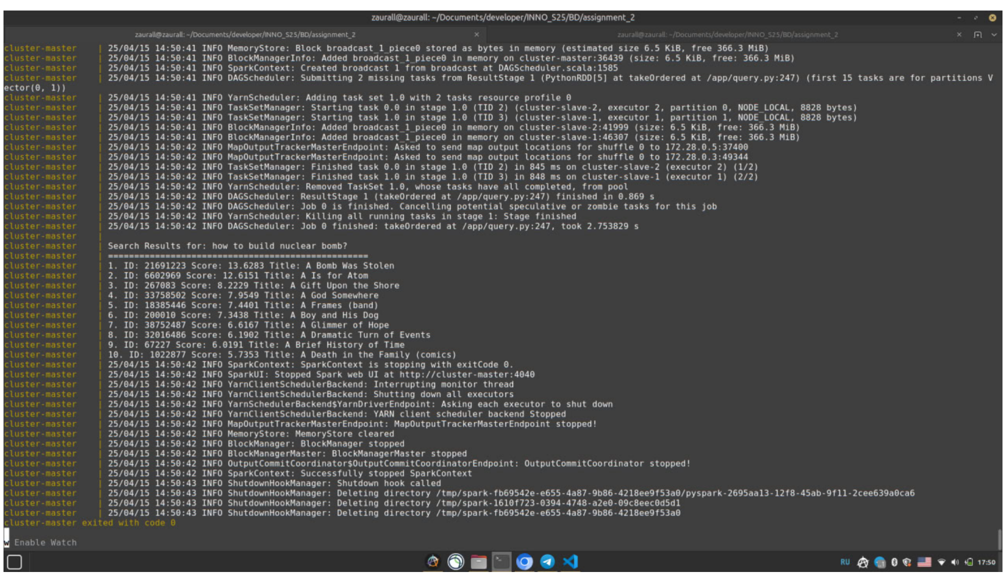

Figure 18: Query 3 results