Big Data Project Report

Course: Big Data

Academic Year: 2025/2026

Project Title:

Search_Engine_Project

Group Name: ITNH

Team Members:

Leonoor Antje Barton

Adrian Budzich

Martyna Chmielińska

Ángela López Dorta

Pablo Mendoza Rodríguez

GitHub Repository:

`https://github.com/InputTheNameHere/stage_1`

Universidad de Las Palmas de Gran Canaria

October 2025

Contents

1. Introduction and Objectives

The goal of the *Search_ Engine_ Project* is to design and implement the **data layer** of a search engine from scratch, as part of Stage 1 of the Big Data course. This layer forms the foundation of a modular data pipeline that can be extended in later stages with indexing, querying, and microservice integration.

The objectives of Stage 1 are:

- Download public domain books from Project Gutenberg.
- Store raw and cleaned text in a well-structured **datalake**.
- Extract metadata and organize it in **datamarts**.
- Evaluate multiple database engines (SQLite, PostgreSQL, MongoDB).
- Coordinate ingestion and indexing operations through a **control layer**.

This stage aims to build a scalable, traceable, and efficient storage architecture that reflects real-world Big Data systems.

2. System Architecture

The system architecture is divided into three layers: the **Datalake**, the **Datamarts**, and the **Control Layer**. Each plays a specific role in managing data flow from ingestion to indexing.

2.1 Datalake

The datalake is designed as a hierarchical repository for storing raw and cleaned text files downloaded from Project Gutenberg.

Structure

```
datalake/
 YYYYMMDD/
     HH/
         <book_id>_header.txt
         <book_id>_body.txt
```

Benefits

- **Traceability:** Each book's ingestion time is recorded through its folder structure.
- **Incremental Processing:** New data can be indexed without reprocessing all files.
- **Scalability:** The structure avoids large single-folder bottlenecks.

2.2 Datamarts

The datamart layer stores structured, queryable information derived from the datalake. It consists of two main components: the **Metadata Datamart** and the **Inverted Index**.

Metadata

Metadata is extracted from book headers and stored in databases for fast querying. The fields include:

- book_id
- title
- author
- language

Database Backends

Three database systems were implemented and tested:

- **SQLite:** Lightweight embedded database.
- **PostgreSQL:** Traditional relational system with indexing and transactions.
- **MongoDB:** NoSQL database supporting flexible documents and scalability.

Schema Example

```
CREATE TABLE books (
    book_id INTEGER PRIMARY KEY,
    title TEXT,
    author TEXT,
    language TEXT
);
```

2.3 Control Layer

The control layer coordinates ingestion and indexing processes through simple tracking files.

```
control/
 downloaded_books.txt
 failed_downloads.txt
 ids_to_download.txt
```

Functions

- Avoid duplicate downloads.
- Schedule new downloads and indexing tasks.
- Update tracking files after each operation.

3. Design Decisions

The chosen data structures and indexing strategies are based on the balance between scalability and query performance.

3.1 Data Structures

- The datalake uses a nested folder hierarchy (date/hour) for scalability.
- Metadata is stored as tables in SQLite/PostgreSQL and documents in MongoDB.
- Each database backend follows the same schema to enable consistent benchmarking.

3.2 Indexing Strategies

Stage 1 focuses on metadata extraction; the inverted index will be developed in Stage 2. However, preliminary design plans include:

- **Monolithic JSON Index:** All terms and postings in one file.
- **MongoDB-based Index:** Terms stored as individual documents.
- **Hierarchical Folder Index:** Each term stored in a separate file under alphabetic folders.

These designs balance simplicity, scalability, and maintainability.

4. Benchmarks and Results

The pipeline was benchmarked on multiple datasets to evaluate its scalability and performance.

Records	Database Engine	Insertion Speed (rows/s)	Average Query Time (s)
10	SQLite	**882.48**	0.000083
	PostgreSQL	152.50	0.000103
	MongoDB	221.12	0.000313
100	SQLite	**2480.06**	0.000065
	PostgreSQL	953.62	0.000078
	MongoDB	1482.02	0.000245
1000	SQLite	**2693.87**	0.000155
	PostgreSQL	2196.34	0.000206
	MongoDB	2204.34	0.001710
10,000	SQLite	545.18	0.016185
	PostgreSQL	**1234.20**	0.027835
	MongoDB	996.99	0.045105
75,393	SQLite	41.24	0.261285
	PostgreSQL	**46.67**	0.205685
	MongoDB	42.58	0.214725

Table 4.1: Performance comparison between SQLite, PostgreSQL, and MongoDB for datasets ranging from 10 to 75393 records.

4.1 Observations

- **SQLite** is fastest for small datasets (<= 1,000 records).
- **PostgreSQL** scales better for large datasets, maintaining higher insertion throughput as the volume grows. It also presents some data loss when big volums of records are involved .
- **MongoDB** offers consistent performance, with query times slightly higher due to network I/O and JSON parsing overhead.
- Performance drops sharply beyond 10,000 rows for all databases, reflecting disk I/O limits and lack of batch optimization.

5. Conclusions and Future Improvements

Stage 1 successfully established a functional and scalable **data layer** for the search engine. The implemented components include:

- A datalake for raw and cleaned text storage.
- Datamarts for structured metadata.
- A control layer for ingestion coordination.

Future Improvements:

- Integrate the inverted index and query modules (Stage 2).
- Implement distributed processing (Spark or multiprocessing).
- Add web interfaces and REST APIs for querying.
- Improve fault tolerance and real-time monitoring.

Bibliography

[1] Project Gutenberg, https://www.gutenberg.org/

[2] SQLite Documentation, https://sqlite.org/docs.html

[3] PostgreSQL Documentation, https://www.postgresql.org/docs/

[4] MongoDB Documentation, https://www.mongodb.com/docs/