DSCI551-Final Project

Crypto Database

Tianzuo Zhang Group39

Demo Video:

https://drive.google.com/file/d/1JniDutMOjKHmDONSyWFRniI0oPB3Yxue/view?usp=drive_link

Code: https://github.com/dvzhang/project551

1. Introduction

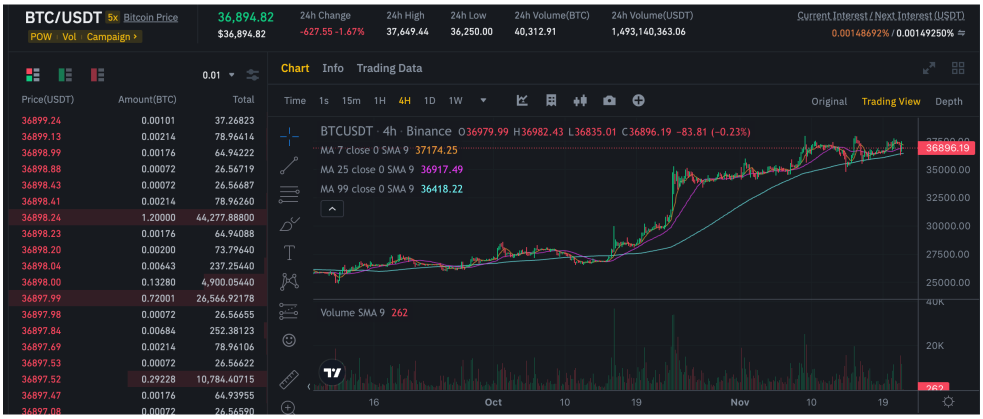
In the contemporary era of data-driven decision-making, the ability to process large volumes of information efficiently is paramount. Our project caters to this need by presenting a system designed to handle extensive datasets using a SQL-like query language, processed through a custom MapReduce framework.

2. Planned Implementation (From Project Proposal)

The cornerstone of our implementation strategy was the chunk-based processing of extensive datasets, a method tailored to overcome the limitations imposed by large file sizes that exceed conventional memory capacities.

Data Acquisition and Chunking:

The project was planned to source its primary dataset from the Binance API, which offers extensive cryptocurrency candle data. The data retrieval mechanism was designed to fetch real-time and historical data, focusing on attributes crucial for in-depth market analysis.



BTC/USDT 5x Bitcoin Price 36,894.82 $36,894.82

24h Change	24h High	24h Low	24h Volume(BTC)	24h Volume(USDT)	Current Interest / Next Interest (USDT)
-627.55 -1.67%	37,649.44	36,250.00	40,312.91	1,493,140,363.06	0.00148692% / 0.00149250%

Price(USDT)	Amount(BTC)	Total
36899.24	0.00101	37.26823
36899.13	0.00214	78.96414
36898.99	0.00176	64.94222
36898.88	0.00072	26.56719
36898.43	0.00072	26.56687
36898.41	0.00214	78.96260
36898.24	1.20000	44,277.88800
36898.23	0.00176	64.94088
36898.20	0.00200	73.79640
36898.04	0.00643	237.25440
36898.00	0.13280	4,900.05440
36897.99	0.72001	26,566.92178
36897.98	0.00072	26.56655
36897.84	0.00684	252.38123
36897.69	0.00214	78.96106
36897.53	0.00072	26.56622
36897.52	0.29228	10,784.40715
36897.47	0.00176	64.93955
36897.08	0.00072	26.56590

BTCUSDT · 4h · Binance O36979.99 H36982.43 L36835.01 C36896.19 -83.81 (-0.23%)

MA 7 close 0 SMA 9 37174.25

MA 25 close 0 SMA 9 36917.49

MA 99 close 0 SMA 9 36418.22

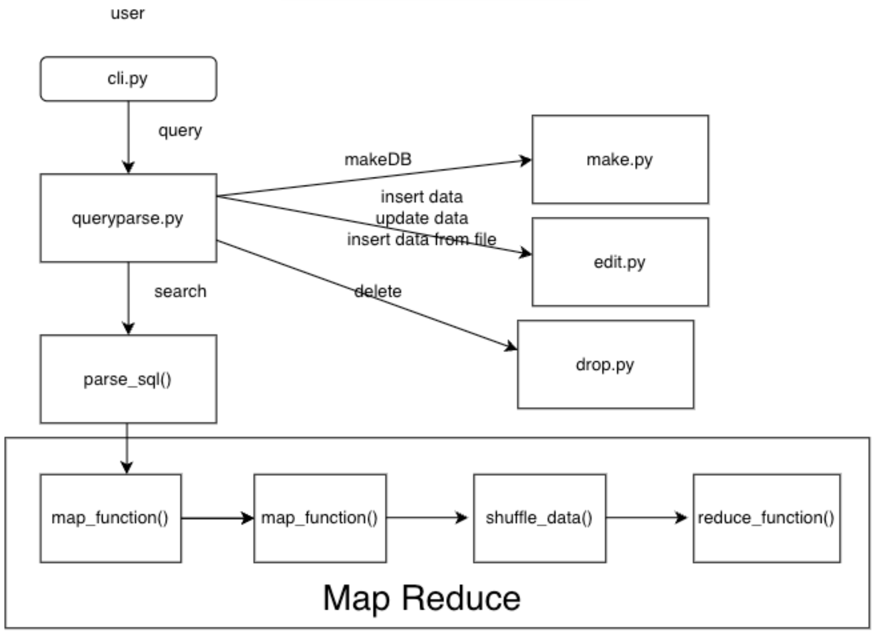
Volume SMA 9 262

Upon acquiring the raw cryptocurrency candle data from the Binance API, the system was designed to partition this data into manageable chunks. This chunking process was a pre-emptive step to ensure that subsequent data processing could be conducted efficiently and without the need for loading entire datasets into memory, thus enabling the handling of data volumes that far exceed typical memory limits.

Chunk-Oriented MapReduce Framework:

The MapReduce framework was specifically architected to operate on these chunks of data. The Map phase was planned to sort and filter within each chunk independently.The Reduce phase was to aggregate results across these chunks, stitching together the partial results into a coherent whole, representative of operations performed on the full dataset.

3. Architecture Design (Flow Diagram and its description)



At the core of the system is a Command Line Interface (CLI), implemented in Python (cli.py), which serves as the primary user interaction point. When a user inputs a command into the CLI, the queryparse function plays a crucial role. It classifies and interprets these commands into distinct categories, such as:

Database Operations: Commands for creating, using, and dropping databases.

Table Manipulations: Commands to create, edit, fetch, or drop tables.

For processing SQL-like queries on large CSV files, the system adopts a MapReduce paradigm, structured as follows:

Map Phase:

Each large CSV file is first split into smaller chunks, making them more manageable for processing.

The system then applies a 'map' operation to each chunk. In this phase, it performs actions like filtering and sorting on individual data records. This step is crucial for preparing the data for the subsequent Reduce phase.

Reduce Phase:

The 'reduce' operation follows the Map phase. It involves aggregating the mapped data.

This stage handles more complex operations such as join, group by, and other aggregation functions like COUNT, AVG, MAX, and MIN.

The outcome of the Reduce phase is a consolidated output that synthesizes the data processed in the Map phase, delivering the final query result.

4. Implementation

The architecture of our MapReduce-based system for processing large CSV files via SQL-like queries can be conceptualized as follows:

a) Command Line Interface (CLI):

The system starts with a CLI, designed in Python, providing an interactive shell

(cli.py) where users can execute various database-related commands.
Users can create, use, or drop databases, as well as perform operations on tables like creating, editing, and fetching data.

b) Query Parsing:

Queries inputted through the CLI are parsed using the parse_query function. This function interprets the user's input and segregates it into distinct commands like MAKEDB, USEDB, DROPDB …...

c) SQL Query Processing (main.py):

For SQL-like data querying, a separate module (main.py) is invoked. This module accepts a modified SQL query, where traditional keywords are replaced (e.g., SELECT with FIND, WHERE with CHARACTER).

The query is then parsed using parse_sql, aligning the syntax with the project's custom SQL parser structure.

d) Data Preprocessing (preprocess.py):

Before processing, large CSV files are split into smaller chunks to facilitate the MapReduce approach.

This is handled by the split_csv function, which divides files like BTC.csv into manageable segments.

e) Map Phase (map.py):

Each chunk of data undergoes the Map phase, where individual records are processed.

Operations like filtering (CHARACTER), projection (FIND), and local sorting (LINE) are applied.

The results are then stored as intermediate sorted files.

f) Shuffle Phase (shuffle.py):

Post Map phase, the Shuffle phase organizes the output for the Reduce phase. It consolidates the sorted intermediate data, preparing it for aggregation and merging.

g) Reduce Phase (reduce.py):

The final phase involves aggregation and joining operations.

Custom functions handle GROUP BY (BUNCH), JOIN (CONNECT), and aggregation functions like count, avg, max, min.

The final output is then generated, aggregating data across all chunks.

5. Functionalities

makedb test2
This command creates a new database named test2.

usedb test2
This command switches the current operation context to the database named test2.

makedb test2
This command is repetitive and if the database test2 already exists, it would typically return an error indicating that the database name is already in use.

```
makedb test3
```

This command creates a new database named test3.

```
showdb
```

This command lists all the databases currently available.

```
dropdb test3
```

This command deletes the database named test3.

```
usedb test2
```

This command switches the context back to the database named test2 to ensure subsequent commands are executed against the correct database.

```
make try COLUMNS a=int, b=str
```

This command creates a new table named try within the current database, with two columns: a with data type integer and b with data type string.

```
make COPY try tryt
```

This command creates a copy of the table named try, with the new table named tryt. This operation copies only the schema of the table, not the data.

```
edit try insert a=1 b="abc"
```

This command inserts a new record into the table named try with the value of column a set to 1 and column b set to "abc".

```
EDIT BTC.csv INSERT FILE BTC.csv
```

This command imports data from a file named BTC.csv into the table structure previously defined for BTC.csv.

```
MAKE BTCALL.csv COLUMNS candle_begin_time=datetime64, ...
```

Similar to the previous MAKE command, this creates another table structure for a file named BTCALL.csv with the same data types as BTC.csv.

```
EDIT BTCALL.csv INSERT FILE BTCALL.csv
```

This command imports data from a file named BTCALL.csv into the table structure previously defined for BTCALL.csv.

```
FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000
```

This command is an SQL-like query that selects records from BTC.csv where the volume is less than 1000. The result will include the columns candle_begin_time, volume, and symbol.

```
FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume
```

< 1000 LINE volume
This command does the same as the previous one but also orders the results by the volume column.

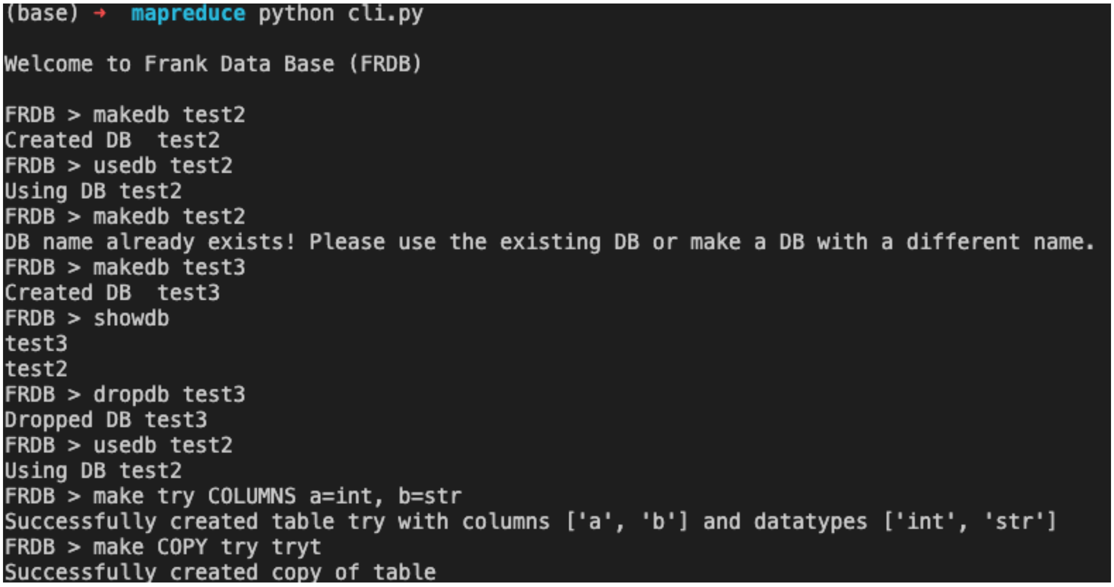
FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000 BUNCH symbol max(volume)
This command fetches records where volume is less than 1000 and groups the results by symbol, calculating the maximum volume for each group.

FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000 CONNECT BTCALL.csv ON symbol
This command performs a join operation between BTC.csv and BTCALL.csv on the symbol column, fetching records where volume is less than 1000 from BTC.csv.

6. Tech Stack
   Python: The entire project is implemented in Python
   Libraries and Frameworks:
   cmd: A Python library used to build the command-line interface (CLI) for our system, providing a user-friendly way to interact with our database management system.
   os: This standard Python library is used for interacting with the operating system, enabling us to handle file and directory operations, which are fundamental to database and table manipulation commands.
   Database Management:
   re (Regular Expressions): Python's built-in library for working with Regular Expressions, which is used for parsing and interpreting custom SQL-like query syntax within our application.

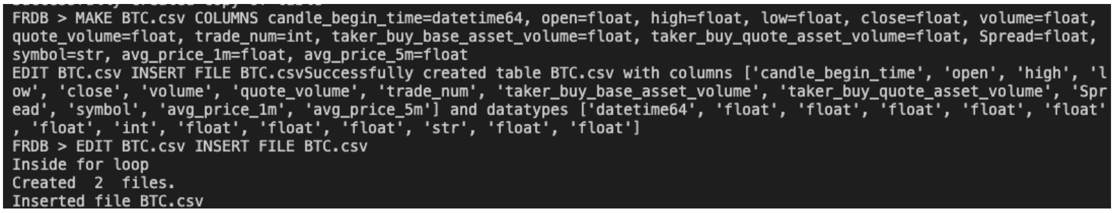
7. Implementation Screenshots (Few not all)

Make datebase / use database / show database / drop database / make new table



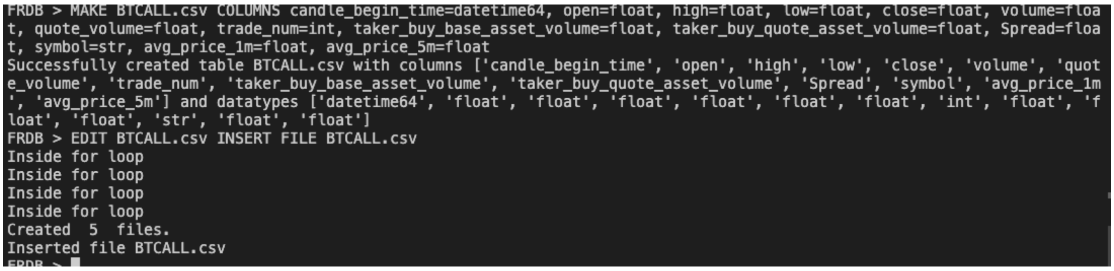
```
(base) ➜  mapreduce python cli.py

Welcome to Frank Data Base (FRDB)

FRDB > makedb test2
Created DB  test2
FRDB > usedb test2
Using DB test2
FRDB > makedb test2
DB name already exists! Please use the existing DB or make a DB with a different name.
FRDB > makedb test3
Created DB  test3
FRDB > showdb
test3
test2
FRDB > dropdb test3
Dropped DB test3
FRDB > usedb test2
Using DB test2
FRDB > make try COLUMNS a=int, b=str
Successfully created table try with columns ['a', 'b'] and datatypes ['int', 'str']
FRDB > make COPY try tryt
Successfully created copy of table
```

Insert a small file (create only one chunk)

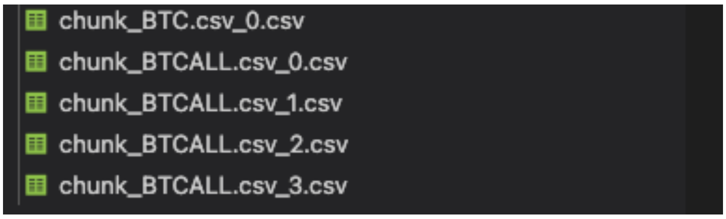


```
FRDB > MAKE BTC.csv COLUMNS candle_begin_time=datetime64, open=float, high=float, low=float, close=float, volume=float,
quote_volume=float, trade_num=int, taker_buy_base_asset_volume=float, taker_buy_quote_asset_volume=float, Spread=float,
symbol=str, avg_price_1m=float, avg_price_5m=float
EDIT BTC.csv INSERT FILE BTC.csvSuccessfully created table BTC.csv with columns ['candle_begin_time', 'open', 'high', 'l
ow', 'close', 'volume', 'quote_volume', 'trade_num', 'taker_buy_base_asset_volume', 'taker_buy_quote_asset_volume', 'Spr
ead', 'symbol', 'avg_price_1m', 'avg_price_5m'] and datatypes ['datetime64', 'float', 'float', 'float', 'float', 'float'
, 'float', 'int', 'float', 'float', 'float', 'str', 'float', 'float']
FRDB > EDIT BTC.csv INSERT FILE BTC.csv
Inside for loop
Created  2  files.
Inserted file BTC.csv
```

Insert a big file (create many chunks)

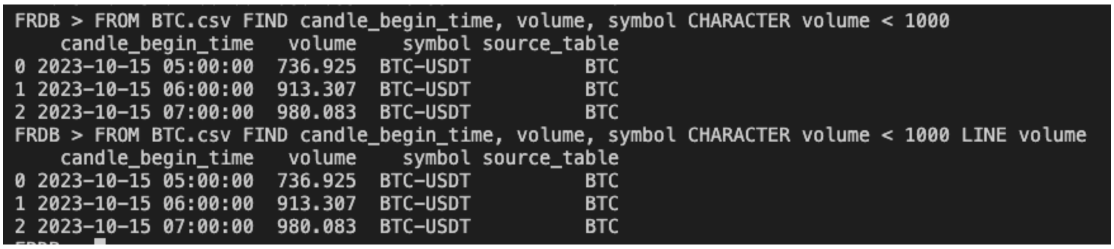


```
FRDB > MAKE BTCALL.csv COLUMNS candle_begin_time=datetime64, open=float, high=float, low=float, close=float, volume=floa
t, quote_volume=float, trade_num=int, taker_buy_base_asset_volume=float, taker_buy_quote_asset_volume=float, Spread=floa
t, symbol=str, avg_price_1m=float, avg_price_5m=float
Successfully created table BTCALL.csv with columns ['candle_begin_time', 'open', 'high', 'low', 'close', 'volume', 'quot
e_volume', 'trade_num', 'taker_buy_base_asset_volume', 'taker_buy_quote_asset_volume', 'Spread', 'symbol', 'avg_price_1m
', 'avg_price_5m'] and datatypes ['datetime64', 'float', 'float', 'float', 'float', 'float', 'float', 'int', 'float', 'f
loat', 'float', 'str', 'float', 'float']
FRDB > EDIT BTCALL.csv INSERT FILE BTCALL.csv
Inside for loop
Inside for loop
Inside for loop
Inside for loop
Created  5  files.
Inserted file BTCALL.csv
FRDB >
```



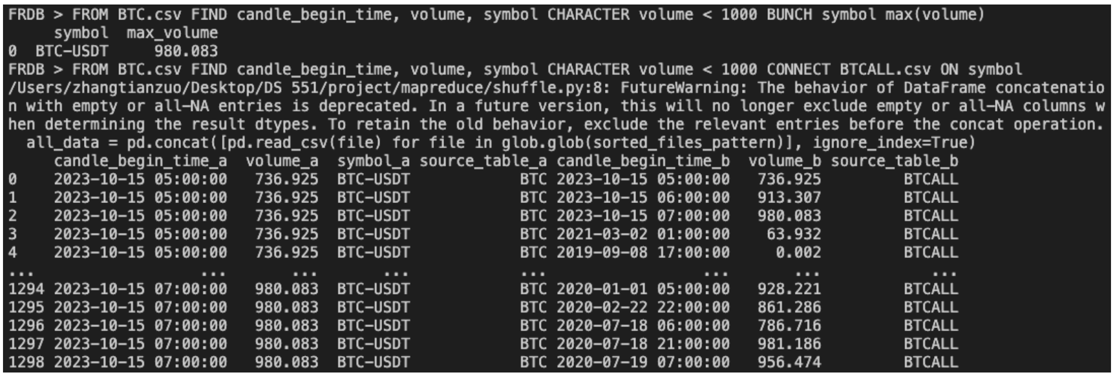
```
chunk_BTC.csv_0.csv
chunk_BTCALL.csv_0.csv
chunk_BTCALL.csv_1.csv
chunk_BTCALL.csv_2.csv
chunk_BTCALL.csv_3.csv
```

Search (select, from, where, order by)



```
FRDB > FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000
    candle_begin_time   volume    symbol source_table
0 2023-10-15 05:00:00  736.925  BTC-USDT          BTC
1 2023-10-15 06:00:00  913.307  BTC-USDT          BTC
2 2023-10-15 07:00:00  980.083  BTC-USDT          BTC
FRDB > FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000 LINE volume
    candle_begin_time   volume    symbol source_table
0 2023-10-15 05:00:00  736.925  BTC-USDT          BTC
1 2023-10-15 06:00:00  913.307  BTC-USDT          BTC
2 2023-10-15 07:00:00  980.083  BTC-USDT          BTC
```

Search (select, from, where, group by,   join)



```
FRDB > FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000 BUNCH symbol max(volume)
     symbol  max_volume
0  BTC-USDT     980.083
FRDB > FROM BTC.csv FIND candle_begin_time, volume, symbol CHARACTER volume < 1000 CONNECT BTCALL.csv ON symbol
/Users/zhangtianzuo/Desktop/DS 551/project/mapreduce/shuffle.py:8: FutureWarning: The behavior of DataFrame concatenatio
n with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns w
hen determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  all_data = pd.concat([pd.read_csv(file) for file in glob.glob(sorted_files_pattern)], ignore_index=True)
     candle_begin_time_a  volume_a  symbol_a source_table_a candle_begin_time_b  volume_b source_table_b
0    2023-10-15 05:00:00   736.925  BTC-USDT            BTC 2023-10-15 05:00:00   736.925         BTCALL
1    2023-10-15 05:00:00   736.925  BTC-USDT            BTC 2023-10-15 06:00:00   913.307         BTCALL
2    2023-10-15 05:00:00   736.925  BTC-USDT            BTC 2023-10-15 07:00:00   980.083         BTCALL
3    2023-10-15 05:00:00   736.925  BTC-USDT            BTC 2021-03-02 01:00:00    63.932         BTCALL
4    2023-10-15 05:00:00   736.925  BTC-USDT            BTC 2019-09-08 17:00:00     0.002         BTCALL
...                  ...       ...       ...            ...                 ...       ...            ...
1294 2023-10-15 07:00:00   980.083  BTC-USDT            BTC 2020-01-01 05:00:00   928.221         BTCALL
1295 2023-10-15 07:00:00   980.083  BTC-USDT            BTC 2020-02-22 22:00:00   861.286         BTCALL
1296 2023-10-15 07:00:00   980.083  BTC-USDT            BTC 2020-07-18 06:00:00   786.716         BTCALL
1297 2023-10-15 07:00:00   980.083  BTC-USDT            BTC 2020-07-18 21:00:00   981.186         BTCALL
1298 2023-10-15 07:00:00   980.083  BTC-USDT            BTC 2020-07-19 07:00:00   956.474         BTCALL
```

8. Learning Outcomes

Advanced Python Proficiency: We have significantly improved our Python coding abilities, particularly in writing complex data processing algorithms that adhere to the MapReduce paradigm.

MapReduce Understanding: Gaining a deep understanding of the MapReduce model was instrumental. We learned how to effectively break down data processing tasks into discrete map and reduce steps, which is crucial for handling large datasets.

System Design: This project has improved our ability to design complex systems that manage large-scale data, from initial input to final output, while ensuring data integrity and system reliability.

9. Challenges Faced

Memory Management: One of the most significant challenges was managing memory while processing large CSV files. We had to ensure that our data processing algorithms were optimized to run efficiently without exceeding memory limits.

Custom MapReduce Logic: Implementing a custom MapReduce framework from scratch was challenging, as it required meticulous attention to detail to ensure that the map, shuffle, and reduce phases worked seamlessly together.

10. Conclusion

In conclusion, this project has achieved its primary goal of developing a robust system capable of processing SQL-like queries on large CSV datasets using a custom MapReduce approach. The successful implementation of the command-line interface and the integration of various Python libraries to handle data manipulation, file operations, and query parsing have proven effective. The project's architecture not only addresses the immediate requirements but also demonstrates the potential for scalability and flexibility.

11. Future Scope

Distributed Processing: To further improve performance on even larger datasets, we could adapt our MapReduce framework to run in a distributed computing environment.

Support for Additional Data Formats: While currently tailored for CSV files, the system could be extended to handle other data formats such as JSON or XML.