

COMP-1702 Big Data
2024-25 Term-2

Tasks Report

Kruthika Mysore Bhaskar
001354599

MSc Data Science

Table of Contents

1. Task A: Hadoop Analysis	1
1.1 Introduction	1
1.2 Genomics Data Processing Using Hadoop.....	1
1.3 The Benefits of Using Hadoop	1
1.4 Conclusion.....	2
2. Task B: Hive Data Warehouse Design	3
2.1 Introduction	3
2.2 Implementation Steps in Hadoop VM	3
2.3 Creating Tables and Load Data in Hive	6
2.4 Executing Queries in Hive	11
2.4.1 Basic Queries.....	11
2.4.2 Analytical Queries	15
3. Task C - MapReduce Programming	26
3.1 Problem Definition	26
3.2 MapReduce Algorithm Design	26
3.2.1 Mapping Stage	26
3.2.2 Mapper Pseudocode:.....	27
3.2.4 Combiner Stage	28
3.2.5 Combiner Pseudocode:.....	28
3.2.6 Stage of Shuffling and Sorting.....	28
3.2.7 Reducing Stage	29
3.2.8 Reducer Pseudocode:	29
3.3 Explanation of the Output	30
3.4 Performance Analysis and Algorithm Efficiency.....	30
4. Task D Big Data Project Analysis – PA Company	32
4.1 Task D.1: Data Storage – Data Warehouse vs. Data Lake	32
4.2 Task D.2: Real-Time Processing – Evaluating MapReduce and Alternatives.....	33
4.3 Task D.3: Cloud Hosting Strategy – Ensuring Security, Scalability, and Availability	34
Reference	35

Table of Figure

Figure 1: Screenshot of starting Hadoop and Yarn	4
Figure 2: Screenshot of Arline_flights Directory	5
Figure 3: Screenshot of all data set files	5
Figure 4 Screenshot of uploading data files to HDFS	6
Figure 5 Screenshot of successful passengers table creation	7
Figure 6 Screenshot of Description of table passengers	7
Figure 7 Screenshot of successful flights table creation	8
Figure 8 Screenshot of Description of table passengers	9
Figure 9 Screenshot of successful bookings table creation	9
Figure 10 Screenshot of Description of table passengers	10
Figure 11 Screenshot of successfully loading data into passengers table.....	10
Figure 12 Screenshot of successfully loading data into flights table	11
Figure 13 Screenshot of successfully loading data into bookings table	11
Figure 14 Screenshot of basic query display passenger table details part-1	12
Figure 15 Screenshot of basic query display passenger table details part-2	13
Figure 16 Screenshot of basic query display flights table details part-1	13
Figure 17 Screenshot of basic query display flights table details part-2	14
Figure 18 Screenshot of basic query display bookings table details part-1.....	14
Figure 19 Screenshot of basic query display bookings table details part-2.....	15
Figure 20 Screenshot of Total number of bookings per flight part-1.....	16
Figure 21 Screenshot of Total number of bookings per flight part-2.....	16
Figure 22 Screenshot of the most popular destination.	17
Figure 23 Screenshot of the most expensive ticket booked	18
Figure 24 Screenshot of the average price of Economy-class tickets	19
Figure 25 Screenshot of passengers list booked flights from USA	20
Figure 26 Screenshot of flights details departing on a specific date (2024-04-01)	21
Figure 27 Screenshot of the busiest flight route (most booked) part -1.....	22
Figure 28 Screenshot of the busiest flight route (most booked) part-2.....	22
Figure 29 Screenshot of detailed Booking Information part-1.....	23
Figure 30 Screenshot of detailed Booking Information part-2.....	23
Figure 31 Screenshot of bookings per Flight part-1	24
Figure 32 Screenshot of bookings per Flight part-2	25
Figure 33 Screenshot of the Most Frequent Routes (Origin → Destination)	25

1. Task A: Hadoop Analysis

A Real-World Scenario: Genomics Data Processing Using Hadoop

1.1 Introduction

The enormous volume and complexity of genomic data produced by Next-Generation Sequencing (NGS) technologies present formidable obstacles. Traditional data systems find it difficult to manage the computational demands of tasks like variant calling and sequence alignment when dataset sizes reach terabytes. Large-scale genomics data can be efficiently managed and processed with Hadoop's distributed storage (HDFS) and parallel processing (MapReduce) features (Shvachko et al., 2010).

1.2 Genomics Data Processing Using Hadoop

In this case, massive genomic datasets are managed using Hadoop. Scalable processing and storage frameworks are needed to manage the growing volume of genomic sequences. Gene expression analysis, variant calling, and sequence alignment are important genomics analysis tasks.

Data Storage: Large genomics datasets can be more easily stored with HDFS's scalable and fault-tolerant storage, which also ensures redundancy and accessibility (Holmes, 2012).

Data processing: By breaking down genomics tasks into smaller subtasks, MapReduce allows for parallel processing, which greatly accelerates analysis (Dean & Ghemawat, 2004).

Resource Management: For large-scale genomics analysis, YARN optimises task execution throughout the Hadoop cluster by dynamically allocating computational resources (Stonebraker et al., 2010).

1.3 The Benefits of Using Hadoop

1. HDFS – Scalable Storage

Because HDFS splits data into blocks and distributes them among cluster nodes, it is ideal for managing big datasets, like those found in genomics. This solves the problems caused by big genomic datasets by guaranteeing scalability and fault tolerance (Shvachko et al., 2010). Because genomic data keeps growing, it is impossible to rely on conventional data storage systems, so this scalability is essential.

2. MapReduce – Parallel Processing

MapReduce makes it possible to process genomics tasks in parallel effectively. Sequence alignment and gene expression analysis are examples of large tasks that can be split up into

smaller subtasks and executed concurrently across multiple nodes. As a result, less computation time is needed, and resources are used as efficiently as possible (Dean & Ghemawat, 2004). Hadoop speeds up data analysis by utilising parallel processing, which is essential for genomics research that needs to be completed quickly.

2. YARN – Efficient Resource Management

According to the demands of the tasks, YARN makes sure that computational resources are distributed dynamically. YARN maximises throughput and ensures efficient processing by optimising resource distribution across the Hadoop cluster, which is necessary for computationally intensive processes in genomics workflows that demand flexible resource management (Stonebraker et al., 2010).

1.4 Conclusion

Hadoop is the perfect framework for handling and processing massive amounts of genomics data because it combines HDFS, MapReduce, and YARN. Its capacity to manage enormous volumes of data through parallel processing, scalable storage, and effective resource management tackles important issues in genomics research. Researchers can advance domains like disease research and personalised medicine by using Hadoop to speed up data analysis.

2. Task B: Hive Data Warehouse Design

2.1 Introduction

Airline operations, as a part of their everyday functioning which includes passenger details, flight schedule, and ticket bookings, produce a huge amount of structured data and records. The proper storage, retrieval, and analysis of this data remain the cornerstone of the airline services optimization and the required data for instant decision-making. This report depicts the utilization of a Hive Data Warehouse, which is a part of the Hadoop Virtual Machine (VM) environment, to do the booking trend analysis, passenger insights, and flight performance analysis.

2.2 Implementation Steps in Hadoop VM

The implementation of the Hive-based Data Warehouse for the Airline Booking System involves several steps, beginning with the justification for its selection, followed by the technical setup and execution of queries.

The use case was based on the Airline Booking System because airlines generate a huge volume of structured data, say flight schedules, passenger details, and bookings. Processing this information can provide valuable business insights, such as identifying peak travel periods, optimizing price levels, and defining passenger trends.

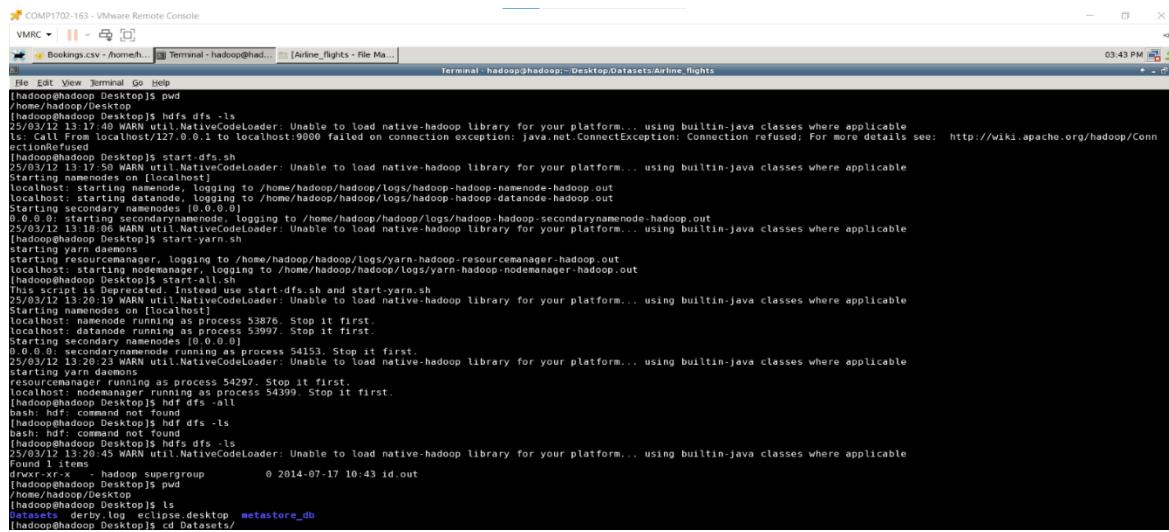
Airline data is also in a good structure, which is favorable to Hive, an SQL-like query language for Hadoop Distributed File System (HDFS). Hadoop's scalability also means that vast volumes of airline data can be processed and stored cost-effectively.

To begin the implementation, “Hadoop services were initialized” using the commands:

i) Starting of Hadoop and Yarn

```
start-dfs.sh          # Starts Hadoop Distributed File System (HDFS)  
start-yarn.sh         # Starts YARN (Resource Management)
```

OUTPUT:



The screenshot shows a terminal window titled 'Terminal - hadoop@had...'. The session is running on a VMware Remote Console. The terminal output displays the execution of Hadoop commands to start the NameNode, DataNodes, and YARN services. The logs indicate several WARN messages related to the NativeCodeLoader failing to load native-hadoop libraries, which is typical for a Java-based ecosystem like Hadoop. The process starts at 03:43 PM on 25/03/12.

```
[hadoop@hadoop Desktop]$ pwd
/home/hadoop/Desktop
[hadoop@hadoop Desktop]$ hdfs dfs -ls
25/03/12 13:47:48 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/03/12 13:47:48 WARN util.NativeCodeLoader: Call from localhost/127.0.0.1 to localhost:9000 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: http://wiki.apache.org/hadoop/ConnectionRefused
[hadoop@hadoop Desktop]$ start-dfs.sh
Starting NameNode on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop/logs/hadoop-namenode-hadoop.out
localhost: starting datanode, logging to /home/hadoop/hadoop/logs/hadoop-datanode-hadoop.out
0.0.0.0: starting secondarynamenode, logging to /home/hadoop/hadoop/logs/hadoop-secondarynamenode-hadoop.out
25/03/12 13:48:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[hadoop@hadoop Desktop]$ start-yarn.sh
starting resourcemanager, logging to /home/hadoop/hadoop/logs/yarn-hadoop-resourcemanager-hadoop.out
localhost: starting nodemanager, logging to /home/hadoop/hadoop/logs/yarn-hadoop-nodemanager-hadoop.out
The input is Dirs... Instead use start-dfs.sh and start-yarn.sh
25/03/12 13:48:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: datanode running as process 53876. Stop it first.
localhost: datanode running as process 53997. Stop it first.
Starting secondary namenodes [0.0.0.0]
Starting secondarynamenode as process 54153. Stop it first.
25/03/12 13:49:09 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
starting yarn daemons
resourcemanager running as process 54297. Stop it first.
localhost: nodemanager running as process 54399. Stop it first.
[hadoop@hadoop Desktop]$ hdfs dfs -ls
[hadoop@hadoop Desktop]$ hdfs dfs -ls
25/03/12 13:50:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
drwxr-xr-x - hadoop supergroup 0 2014-07-17 10:43 id.out
[hadoop@hadoop Desktop]$ pwd
/home/hadoop/Desktop
[hadoop@hadoop Desktop]$ ls
[hadoop@hadoop Desktop]$ ls
[hadoop@hadoop Desktop]$ cd Datasets
[hadoop@hadoop Desktop]$ cd Datasets/
```

Figure 1: Screenshot of starting Hadoop and Yarn

HDFS serves as the primary storage for airline data, while YARN manages resource allocation for efficient query execution, Fig 1 illustrates the execution of start Hadoop and yarn resource management. The output confirms that the Hadoop ecosystem, such as NameNode, DataNode and Resource Manager have started successfully and now are connected, enabling distributed data processing.

ii) Creating CSV

Structured data was prepared in CSV format and uploaded to HDFS. The data includes:

- Passengers.csv – Passenger details.
- Flights.csv – Flights schedules and airline details.
- Bookings.csv – Ticket bookings.

Once the data was uploaded, ‘Hive tables were created’ to store the information in a structured format. The tables include ‘Passengers, Flights, and Bookings’, each defined with appropriate data types to facilitate query execution.

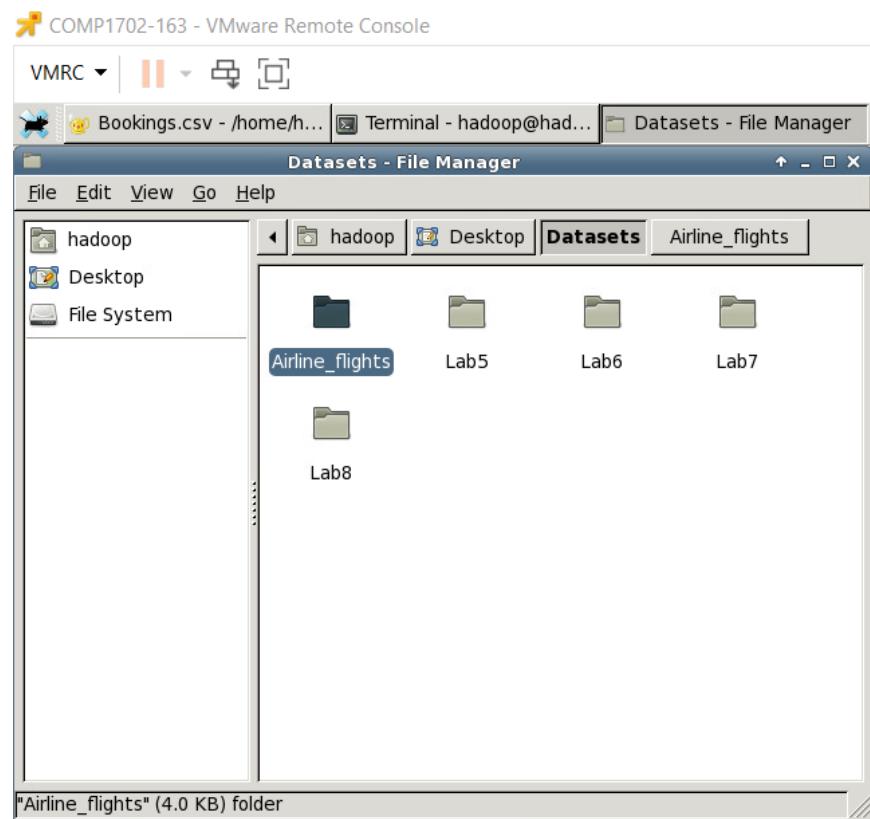


Figure 2: Screenshot of Arline_flights Directory

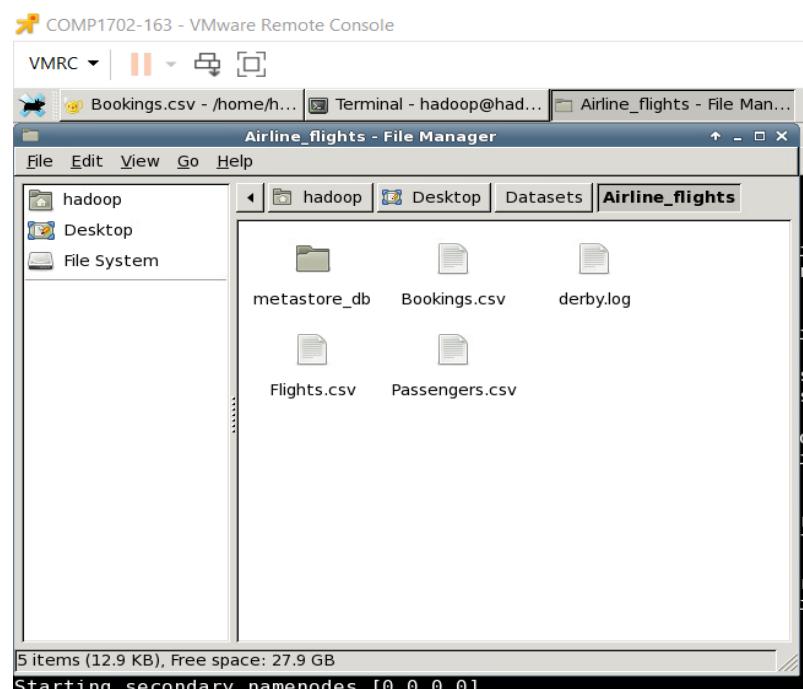


Figure 3: Screenshot of all data set files

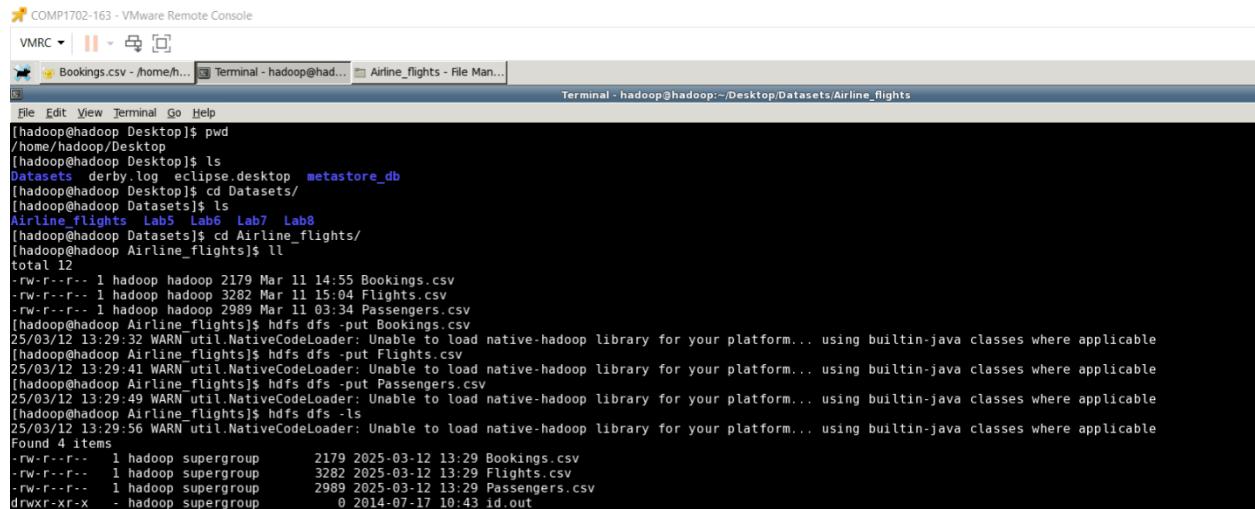
iii) Uploading Data to HDFS

Before creating the tables in Hive, the prepared CSV files are transferred to Hadoop Distributed File System (HDFS) using the following commands:

Input

hdfs dfs -put Passengers.csv	#Uploading Passangers.csv to HDFS
hdfs dfs -put Bookings.csv	#Uploading Bookings.csv to HDFS
hdfs dfs -put Flights.csv	#Uploading Flights.csv to HDFS

OUTPUT:



The screenshot shows a terminal window titled 'Terminal - hadoop@hadoop:~/Desktop/Datasets/Airline_flights'. The window displays the following command-line session:

```
[hadoop@hadoop Desktop]$ pwd
/home/hadoop/Desktop
[hadoop@hadoop Desktop]$ ls
Datasets derby.log eclipse.desktop metastore_db
[hadoop@hadoop Desktop]$ cd Datasets/
[hadoop@hadoop Datasets]$ ls
Airline_flights Lab5 Lab6 Lab7 Lab8
[hadoop@hadoop Datasets]$ cd Airline_flights/
[hadoop@hadoop Airline_flights]$ ll
total 12
-rw-r--r-- 1 hadoop hadoop 2179 Mar 11 14:55 Bookings.csv
-rw-r--r-- 1 hadoop hadoop 3282 Mar 11 15:04 Flights.csv
-rw-r--r-- 1 hadoop hadoop 2989 Mar 11 03:34 Passengers.csv
[hadoop@hadoop Airline_flights]$ hdfs dfs -put Bookings.csv
25/03/12 13:29:32 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[hadoop@hadoop Airline_flights]$ hdfs dfs -put Flights.csv
25/03/12 13:29:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[hadoop@hadoop Airline_flights]$ hdfs dfs -put Passengers.csv
25/03/12 13:29:49 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[hadoop@hadoop Airline_flights]$ hdfs dfs -ls
25/03/12 13:29:56 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 4 items
-rw-r--r-- 1 hadoop supergroup 2179 2025-03-12 13:29 Bookings.csv
-rw-r--r-- 1 hadoop supergroup 3282 2025-03-12 13:29 Flights.csv
-rw-r--r-- 1 hadoop supergroup 2989 2025-03-12 13:29 Passengers.csv
drwxr-xr-x - hadoop supergroup 0 2014-07-17 10:43 id.out
```

Figure 4 Screenshot of uploading data files to HDFS

Fig 4 depicts uploading of the datasets. These commands ensure that the datasets are stored in HDFS, making them accessible for Hive table creation and further analysis.

Once the data was uploaded, Hive tables will be created to store the information in a structured format. The tables include Passengers, Flights, and Bookings, each defined with appropriate data types to facilitate query execution. Verified datasets Using 'hdfs dfs -ls'.

2.3 Creating Tables and Load Data in Hive

i) Passengers Table Creation

```
CREATE TABLE passengers (
    passenger_id INT,
    name STRING,
    email STRING,
    age INT,
    country STRING
```

```
) ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
STORED AS TEXTFILE  
TBLPROPERTIES ("skip.header.line.count"="1");
```

OUTPUT

```
> CREATE TABLE passengers (  
>     passenger_id INT,  
>     name STRING,  
>     email STRING,  
>     age INT,  
>     country STRING  
> ) ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> STORED AS TEXTFILE  
> TBLPROPERTIES ("skip.header.line.count"="1");  
OK  
Time taken: 0.467 seconds  
hive>  
>
```

Figure 5 Screenshot of successful passengers table creation

Fig 5 shows the query execution which created a Hive table named passengers with five columns (passenger_id, name, email, age, country). The data is stored as a text file, with fields separated by commas (CSV format). The table also skips the first row (header) while reading data. It also displays successful creation by giving ‘OK’ in output.

ii) Describing Table

```
Describe table passengers;
```

OUTPUT

```
> describe passengers;  
OK  
passenger_id          int  
name                  string  
email                 string  
age                  int  
country              string  
Time taken: 0.392 seconds, Fetched: 5 row(s)  
hive>  
>
```

Figure 6 Screenshot of Description of table passengers

iii) Flights Table Creation

```
CREATE TABLE flights (
    flight_id INT,
    airline STRING,
    origin STRING,
    destination STRING,
    departure_time TIMESTAMP,
    arrival_time TIMESTAMP
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

OUTPUT

```
hive>
>
>
> CREATE TABLE flights (
>     flight_id INT,
>     airline STRING,
>     origin STRING,
>     destination STRING,
>     departure_time TIMESTAMP,
>     arrival_time TIMESTAMP
> ) ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE
> TBLPROPERTIES ("skip.header.line.count"="1");
OK
```

Figure 7 Screenshot of successful flights table creation

Fig 7 shows the query execution which creates a Hive table named flights with columns for flight details like flight_id, airline, origin, destination, departure_time, and arrival_time. The data is stored in CSV format, with fields separated by commas. It also displays successful creation by giving 'OK' in output.

iv) Describing Table

```
Describe table flights;
```

OUTPUT

```
hive> > describe flights;
OK
flight_id          int
airline            string
origin             string
destination        string
departure_time     timestamp
arrival_time       timestamp
Time taken: 0.132 seconds, Fetched: 6 row(s)
hive>
>
```

Figure 8 Screenshot of Description of table passengers

iv) Bookings Table Creation

```
CREATE TABLE bookings (
  booking_id INT,
  passenger_id INT,
  flight_id INT,
  booking_date DATE,
  seat_class STRING,
  price DOUBLE
) ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

OUTPUT

```
hive>
> CREATE TABLE bookings (
>   booking_id INT,
>   passenger_id INT,
>   flight_id INT,
>   booking_date DATE,
>   seat_class STRING,
>   price DOUBLE
> ) ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE
> TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.066 seconds
hive>
```

Figure 9 Screenshot of successful bookings table creation

Fig 9 demonstrates the query execution which created table named bookings to store flight booking data with columns like booking_id, passenger_id, flight_id, booking_date, seat_class, and price. The data is stored in comma-separated (CSV) format as a TEXTFILE, and the first line is skipped if it contains a header. This structure is suitable for analyzing booking trends and pricing in large datasets.

v) Describing Table

```
Describe bookings;  
  
OUTPUT  
hive>  
    > describe bookings;  
OK  
booking_id          int  
passenger_id        int  
flight_id           int  
booking_date        date  
seat_class          string  
price               double  
Time taken: 0.133 seconds, Fetched: 6 row(s)  
hive>  
    >
```

Figure 10 Screenshot of Description of table passengers

vi) Loading Data into Hive Tables

Below steps ensured that the airline data was successfully integrated into the Hive Data Warehouse, making it accessible for query execution and analysis.

- ➔ Loading data from Passengers.csv to passengers table

```
LOAD DATA INPATH 'Passengers.csv' OVERWRITE INTO TABLE passengers;  
  
OUTPUT  
    > LOAD DATA INPATH 'Passengers.csv' OVERWRITE INTO TABLE passengers;  
Loading data to table default.passengers  
rmr: DEPRECATED: Please use 'rm -r' instead.  
Deleted hdfs://localhost:9000/user/hive/warehouse/passengers  
Table default.passengers stats: [numFiles=1, numRows=0, totalSize=2989, rawDataSize=0]  
OK  
Time taken: 0.561 seconds  
hive>  
    >
```

Figure 11 Screenshot of successfully loading data into passengers table

➔ Loading data from Flights.csv to flights table

```
LOAD DATA INPATH 'Flights.csv' OVERWRITE INTO TABLE flights;
```

OUTPUT

```
hive>
>
> LOAD DATA INPATH 'Flights.csv' OVERWRITE INTO TABLE flights;
Loading data to table default.flights
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted hdfs://localhost:9000/user/hive/warehouse/flights
Table default.flights stats: [numFiles=1, numRows=0, totalSize=3282, rawDataSize=0]
OK
Time taken: 0.241 seconds
hive>
>
```

Figure 12 Screenshot of successfully loading data into flights table

➔ Loading data from Bookings.csv to bookings table

```
LOAD DATA INPATH 'Bookings.csv' OVERWRITE INTO TABLE bookings;
```

OUTPUT

```
hive>
>
>
> LOAD DATA INPATH 'Bookings.csv' OVERWRITE INTO TABLE bookings;
Loading data to table default.bookings
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted hdfs://localhost:9000/user/hive/warehouse/bookings
Table default.bookings stats: [numFiles=1, numRows=0, totalSize=2179, rawDataSize=0]
OK
Time taken: 0.247 seconds
hive>
>
```

Figure 13 Screenshot of successfully loading data into bookings table

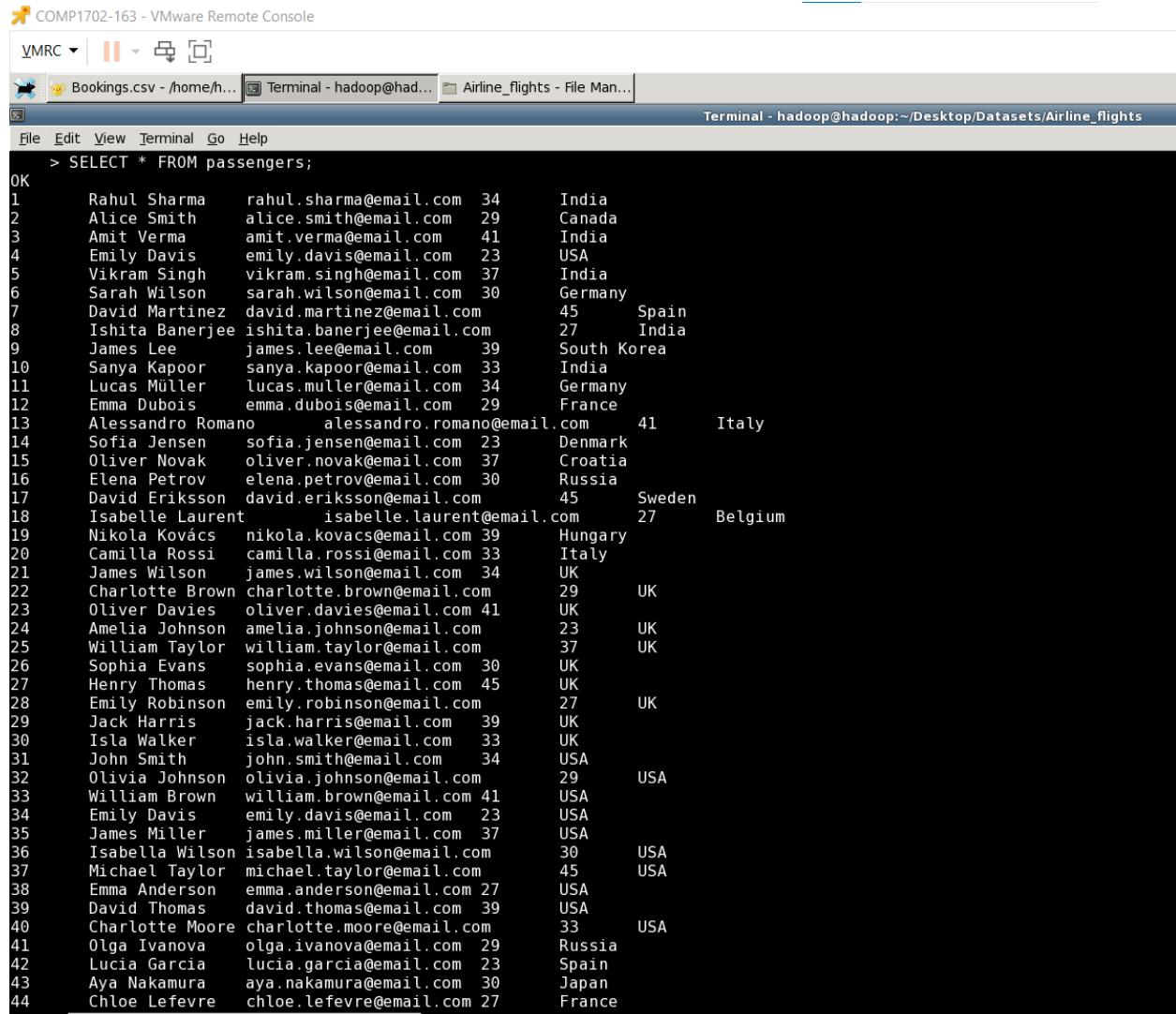
2.4 Executing Queries in Hive

2.4.1 Basic Queries

i) passenger table data display

```
SELECT * FROM passengers;
```

OUTPUT



The screenshot shows a VMware Remote Console window titled 'COMP1702-163 - VMware Remote Console'. Inside, a terminal window is open with the command 'Terminal - hadoop@hadoop:~/Desktop/Datasets/Airline_flights'. The terminal displays the output of a SQL query: 'SELECT * FROM passengers;'. The output lists 44 passengers with their names, email addresses, ages, and countries.

Passenger ID	Name	Email	Age	Country
1	Rahul Sharma	rahul.sharma@email.com	34	India
2	Alice Smith	alice.smith@email.com	29	Canada
3	Amit Verma	amit.verma@email.com	41	India
4	Emily Davis	emily.davis@email.com	23	USA
5	Vikram Singh	vikram.singh@email.com	37	India
6	Sarah Wilson	sarah.wilson@email.com	30	Germany
7	David Martinez	david.martinez@email.com	45	Spain
8	Ishita Banerjee	ishita.banerjee@email.com	27	India
9	James Lee	james.lee@email.com	39	South Korea
10	Sanya Kapoor	sanya.kapoor@email.com	33	India
11	Lucas Müller	lucas.mueller@email.com	34	Germany
12	Emma Dubois	emma.dubois@email.com	29	France
13	Alessandro Romano	alessandro.romano@email.com	41	Italy
14	Sofia Jensen	sofia.jensen@email.com	23	Denmark
15	Oliver Novak	oliver.novak@email.com	37	Croatia
16	Elena Petrov	elena.petrov@email.com	30	Russia
17	David Eriksson	david.eriksson@email.com	45	Sweden
18	Isabelle Laurent	isabelle.laurent@email.com	27	Belgium
19	Nikola Kovács	nikola.kovacs@email.com	39	Hungary
20	Camilla Rossi	camilla.rossi@email.com	33	Italy
21	James Wilson	james.wilson@email.com	34	UK
22	Charlotte Brown	charlotte.brown@email.com	29	UK
23	Oliver Davies	oliver.davies@email.com	41	UK
24	Amelia Johnson	amelia.johnson@email.com	23	UK
25	William Taylor	william.taylor@email.com	37	UK
26	Sophia Evans	sophia.evans@email.com	30	UK
27	Henry Thomas	henry.thomas@email.com	45	UK
28	Emily Robinson	emily.robinson@email.com	27	UK
29	Jack Harris	jack.harris@email.com	39	UK
30	Isla Walker	isla.walker@email.com	33	UK
31	John Smith	john.smith@email.com	34	USA
32	Olivia Johnson	olivia.johnson@email.com	29	USA
33	William Brown	william.brown@email.com	41	USA
34	Emily Davis	emily.davis@email.com	23	USA
35	James Miller	james.miller@email.com	37	USA
36	Isabella Wilson	isabella.wilson@email.com	30	USA
37	Michael Taylor	michael.taylor@email.com	45	USA
38	Emma Anderson	emma.anderson@email.com	27	USA
39	David Thomas	david.thomas@email.com	39	USA
40	Charlotte Moore	charlotte.moore@email.com	33	USA
41	Olga Ivanova	olga.ivanova@email.com	29	Russia
42	Lucia Garcia	lucia.garcia@email.com	23	Spain
43	Aya Nakamura	aya.nakamura@email.com	30	Japan
44	Chloe Lefevre	chloe.lefevre@email.com	27	France

Figure 14 Screenshot of basic query display passenger table details part-1

```

41      Olga Ivanova    olga.ivanova@email.com 29      Russia
42      Lucia Garcia    lucia.garcia@email.com 23      Spain
43      Aya Nakamura    aya.nakamura@email.com 30      Japan
44      Chloe Lefevre   chloe.lefevre@email.com 27      France
45      Kofi Owusu      kofi.owusu@email.com 33      Ghana
46      Ahmed Ali        ahmed.ali@email.com 38      Egypt
47      Leila Bouzid     leila.bouzid@email.com 26      Tunisia
48      Daniela Rodríguez daniela.rodriguez@email.com 35      Mexico
49      Martina Novak    martina.novak@email.com 32      Czech Republic
50      Felipe Silva     felipe.silva@email.com 41      Brazil
51      Amir Khan        amir.khan@email.com 33      Pakistan
52      Claire O'Connor claire.oconnor@email.com 28      Ireland
53      Sarah Ibrahim    sarah.ibrahim@email.com 40      Algeria
54      Rafael Perez    rafael.perez@email.com 38      Argentina
55      Yassir Ben Ali    yassir.benali@email.com 36      Tunisia
56      Jessica Taylor   jessica.taylor@email.com 31      USA
57      Leandro Silva   leandro.silva@email.com 28      Brazil
58      Marco González  marco.gonzalez@email.com 34      Spain
59      Maria Kovács    maria.kovacs@email.com 26      Hungary
60      Johann Schmidt   johann.schmidt@email.com 39      Germany
Time taken: 0.296 seconds, Fetched: 60 row(s)
hive>
>
```

Figure 15 Screenshot of basic query display passenger table details part-2

Fig 14 and Fig 15 displays basic select statement that retrieves all records in the passengers table, displaying all the fields (i.e., passenger ID, name, email, age, and country). It serves as a foundation for presenting the complete data set of passengers available in the system.

ii) flights table data display

```
SELECT * FROM flights;
```

OUTPUT

```

OK
101      American Airlines    JFK      LAX      2024-03-10 08:00:00      2024-03-10 11:30:00
102      Delta Airlines      ORD      MIA      2024-03-11 09:15:00      2024-03-11 13:00:00
103      United Airlines     LAX      DFW      2024-03-12 07:45:00      2024-03-12 11:10:00
104      British Airways    LHR      JFK      2024-03-13 15:30:00      2024-03-13 22:45:00
105      Air Canada         YYZ      YYZ      2024-03-14 06:00:00      2024-03-14 10:15:00
106      Eurowings          DXB      SYD      2024-03-15 12:00:00      2024-03-15 22:30:00
107      Qatar Airways      DOH      BKK      2024-03-16 14:00:00      2024-03-16 20:45:00
108      Lufthansa           FRA      SFO      2024-03-17 10:00:00      2024-03-17 18:20:00
109      Singapore Airlines SIN      NRT      2024-03-18 16:00:00      2024-03-18 22:35:00
110      Turkish Airlines   IST      JNB      2024-03-19 23:00:00      2024-03-20 07:45:00
111      KLM                AMS      CDG      2024-03-20 07:30:00      2024-03-20 09:15:00
112      Air France          CDG      AMS      2024-03-21 18:10:00      2024-03-21 19:45:00
113      Swiss Air           ZRH      LHR      2024-03-22 13:00:00      2024-03-22 14:45:00
114      Qantas              SYD      MEL      2024-03-23 23:00:00      2024-03-24 01:25:00
115      Aeroflot             SVO      HEL      2024-03-24 10:00:00      2024-03-24 14:15:00
116      Jet Airways          DEL      BHM      2024-03-25 06:00:00      2024-03-25 08:30:00
117      Air Berlin            DEL      MAA      2024-03-26 20:00:00      2024-03-27 01:10:00
118      EasyJet LGW          BCN      2024-03-27 11:00:00      2024-03-27 13:00:00
119      Flybe EXT             MAN      2024-03-28 05:15:00      2024-03-28 07:30:00
120      Ryanair DUB          STN      2024-03-29 08:30:00      2024-03-29 09:50:00
121      Jetstar Airways     MEL      BNE      2024-03-30 17:00:00      2024-03-30 19:00:00
122      Alitalia             FCO      LIN      2024-03-31 12:30:00      2024-03-31 14:00:00
123      China Airlines       TPE      HKG      2024-04-01 01:30:00      2024-04-01 05:30:00
124      Emirates             DXB      BOM      2024-04-02 18:00:00      2024-04-02 23:55:00
125      Finnair              HEL      STO      2024-04-03 10:45:00      2024-04-03 13:00:00
126      Turkish Airlines    IST      SGN      2024-04-04 09:00:00      2024-04-04 12:15:00
127      Virgin Atlantic     LHR      JFK      2024-04-05 16:00:00      2024-04-05 20:10:00
128      United Airlines      EWR      SFO      2024-04-06 07:00:00      2024-04-06 10:40:00
129      Lufthansa             FRA      ORD      2024-04-07 14:30:00      2024-04-07 18:45:00
130      American Airlines   DFW      ORD      2024-04-08 06:15:00      2024-04-08 08:50:00
131      Qatar Airways       DOH      SYD      2024-04-09 22:00:00      2024-04-10 10:00:00
132      Jet Airways          DEL      BLR      2024-04-10 15:30:00      2024-04-10 17:45:00
133      Air New Zealand     AKL      SYD      2024-04-11 11:30:00      2024-04-11 13:40:00
134      Emirates             DXB      LHR      2024-04-12 18:00:00      2024-04-13 05:25:00
135      KLM                AMS      BCN      2024-04-13 12:00:00      2024-04-13 14:10:00
136      Swiss Air           ZRH      LAX      2024-04-14 06:30:00      2024-04-14 09:00:00
137      Alitalia             DEL      BHM      2024-04-15 06:00:00      2024-04-15 10:30:00
138      Aeroflot             SVO      LED      2024-04-16 05:45:00      2024-04-16 08:15:00
139      Turkish Airlines   IST      SGN      2024-04-17 21:00:00      2024-04-18 01:30:00
140      Lufthansa             FRA      CDG      2024-04-18 08:00:00      2024-04-18 10:25:00
141      United Airlines     SFO      ORD      2024-04-19 07:10:00      2024-04-19 12:45:00
142      Delta Airlines       ATL      LAX      2024-04-20 09:00:00      2024-04-20 12:10:00
143      Aeroflot             LED      HEL      2024-04-21 16:00:00      2024-04-21 18:30:00
144      Jet Airways          BOM      BLR      2024-04-22 23:30:00      2024-04-23 01:10:00

```

Figure 16 Screenshot of basic query display flights table details part-1

flight_id	origin	destination	airline	origin_time	arrival_time	duration
129	Lufthansa	FRA	ORD	2024-04-07 14:30:00	2024-04-07 18:45:00	
130	American Airlines	DFW	ORD	2024-04-08 06:15:00	2024-04-08 08:50:00	
131	Qatar Airways	DOH	SYD	2024-04-09 22:00:00	2024-04-10 10:00:00	
132	Jet Airways	DEL	BLR	2024-04-10 15:30:00	2024-04-10 17:45:00	
133	Air New Zealand	AKL	SYD	2024-04-11 11:30:00	2024-04-11 13:40:00	
134	Emirates	DXB	LHR	2024-04-12 18:00:00	2024-04-13 05:25:00	
135	KLM	AMS	BCN	2024-04-13 12:00:00	2024-04-13 14:10:00	
136	Swiss Air	ZRH	LAX	2024-04-14 06:30:00	2024-04-14 09:00:00	
137	Air India	DEL	BLR	2024-04-15 08:00:00	2024-04-15 10:30:00	
138	Aeroflot	SVO	LED	2024-04-16 05:45:00	2024-04-16 08:15:00	
139	Turkish Airlines	IST	SGN	2024-04-17 21:00:00	2024-04-18 01:30:00	
140	Lufthansa	FRA	CDG	2024-04-18 08:00:00	2024-04-18 10:25:00	
141	United Airlines	SFO	ORD	2024-04-19 07:10:00	2024-04-19 12:45:00	
142	Delta Airlines	ATL	LAX	2024-04-20 09:00:00	2024-04-20 12:10:00	
143	Aeroflot	LED	HEL	2024-04-21 16:00:00	2024-04-21 18:30:00	
144	Jet Airways	BOM	BLR	2024-04-22 23:30:00	2024-04-23 01:10:00	
145	Virgin Atlantic	LHR	SFO	2024-04-23 10:30:00	2024-04-23 13:45:00	
146	Alitalia	FCO	CDG	2024-04-24 07:15:00	2024-04-24 09:20:00	
147	China Southern Airlines	CAN	BKK	2024-04-25 08:00:00	2024-04-25 12:30:00	
148	Jetstar Airways	BNE	MEL	2024-04-26 17:30:00	2024-04-26 19:00:00	
149	Finnair	HEL	AMS	2024-04-27 06:45:00	2024-04-27 08:15:00	
150	Turkish Airlines	IST	CAI	2024-04-28 10:00:00	2024-04-28 12:20:00	

Figure 17 Screenshot of basic query display flights table details part-2

Fig 16 and Fig 17 display the details of each flight, including the flight_id, airline, origin, destination, departure date, and arrival date. The query is employed to see all the flight information available.

iii) bookings table data display

```
SELECT * bookings;
```

OUTPUT

Terminal - hadoop@hadoop:~/Desktop/Datasets/Airline_Flights\$					
<pre>File Edit View Terminal Go Help</pre>					
<pre>> SELECT * FROM bookings;</pre>					
<pre>OK</pre>					
<pre>1001 1 101 2024-03-01 Economy 250.0</pre>					
<pre>1002 2 102 2024-03-02 Business 450.0</pre>					
<pre>1003 3 103 2024-03-03 Economy 300.0</pre>					
<pre>1004 4 104 2024-03-04 Economy 350.0</pre>					
<pre>1005 5 105 2024-03-05 Business 500.0</pre>					
<pre>1006 6 106 2024-03-06 Economy 400.0</pre>					
<pre>1007 7 107 2024-03-07 Business 550.0</pre>					
<pre>1008 8 108 2024-03-08 Economy 320.0</pre>					
<pre>1009 9 109 2024-03-09 Economy 280.0</pre>					
<pre>1010 10 110 2024-03-10 Business 600.0</pre>					
<pre>1011 11 111 2024-03-11 Economy 230.0</pre>					
<pre>1012 12 112 2024-03-12 Business 550.0</pre>					
<pre>1013 13 113 2024-03-13 Economy 370.0</pre>					
<pre>1014 14 114 2024-03-14 Business 280.0</pre>					
<pre>1015 15 115 2024-03-15 Business 520.0</pre>					
<pre>1016 16 116 2024-03-16 Economy 290.0</pre>					
<pre>1017 17 117 2024-03-17 Business 530.0</pre>					
<pre>1018 18 118 2024-03-18 Economy 300.0</pre>					
<pre>1019 19 119 2024-03-19 Economy 310.0</pre>					
<pre>1020 20 120 2024-03-20 Business 600.0</pre>					
<pre>1021 21 121 2024-03-21 Economy 250.0</pre>					
<pre>1022 22 122 2024-03-22 Business 450.0</pre>					
<pre>1023 23 123 2024-03-23 Economy 350.0</pre>					
<pre>1024 24 124 2024-03-24 Economy 280.0</pre>					
<pre>1025 25 125 2024-03-25 Business 550.0</pre>					
<pre>1026 26 126 2024-03-26 Economy 400.0</pre>					
<pre>1027 27 127 2024-03-27 Business 600.0</pre>					
<pre>1028 28 128 2024-03-28 Economy 300.0</pre>					
<pre>1029 29 129 2024-03-29 Business 520.0</pre>					
<pre>1030 30 130 2024-03-30 Economy 280.0</pre>					
<pre>1031 31 131 2024-03-31 Business 540.0</pre>					
<pre>1032 32 132 2024-04-01 Economy 330.0</pre>					
<pre>1033 33 133 2024-04-02 Economy 320.0</pre>					
<pre>1034 34 134 2024-04-03 Business 620.0</pre>					
<pre>1035 35 135 2024-04-04 Economy 280.0</pre>					
<pre>1036 36 136 2024-04-05 Business 550.0</pre>					
<pre>1037 37 137 2024-04-06 Economy 300.0</pre>					
<pre>1038 38 138 2024-04-07 Economy 250.0</pre>					
<pre>1039 39 139 2024-04-08 Business 530.0</pre>					
<pre>1040 40 140 2024-04-09 Economy 310.0</pre>					
<pre>1041 41 141 2024-04-10 Economy 330.0</pre>					
<pre>1042 42 142 2024-04-11 Business 560.0</pre>					
<pre>1043 43 143 2024-04-12 Economy 270.0</pre>					
<pre>1044 44 144 2024-04-13 Business 500.0</pre>					

Figure 18 Screenshot of basic query display bookings table details part-1

```

1031 31 131 2024-03-31 Business 540.0
1032 32 132 2024-04-01 Economy 330.0
1033 33 133 2024-04-02 Economy 320.0
1034 34 134 2024-04-03 Business 620.0
1035 35 135 2024-04-04 Economy 280.0
1036 36 136 2024-04-05 Business 550.0
1037 37 137 2024-04-06 Economy 300.0
1038 38 138 2024-04-07 Economy 250.0
1039 39 139 2024-04-08 Business 530.0
1040 40 140 2024-04-09 Economy 310.0
1041 41 141 2024-04-10 Economy 330.0
1042 42 142 2024-04-11 Business 560.0
1043 43 143 2024-04-12 Economy 270.0
1044 44 144 2024-04-13 Business 500.0
1045 45 145 2024-04-14 Economy 280.0
1046 46 146 2024-04-15 Business 550.0
1047 47 147 2024-04-16 Economy 320.0
1048 48 148 2024-04-17 Economy 330.0
1049 49 149 2024-04-18 Business 620.0
1050 50 150 2024-04-19 Economy 270.0
1051 51 101 2024-04-20 Business 570.0
1052 52 102 2024-04-21 Economy 240.0
1053 53 103 2024-04-22 Economy 290.0
1054 54 104 2024-04-23 Business 600.0
1055 55 105 2024-04-24 Economy 310.0
1056 56 106 2024-04-25 Business 550.0
1057 57 107 2024-04-26 Economy 250.0
1058 58 108 2024-04-27 Business 530.0
1059 59 109 2024-04-28 Economy 300.0
1060 60 110 2024-04-29 Economy 320.0
Time taken: 0.065 seconds, Fetched: 60 row(s)
hive>
>
>
```

Figure 19 Screenshot of basic query display bookings table details part-2

Fig 18 and Fig 19 display the details of each bookings, including the booking_id, passenger_id, flight_id, booking_date, seat_class, and price. The query sucessfully gives the informations of flight bookings available.

2.4.2 Analytical Queries

i) Total number of bookings per flight

```

SELECT flight_id, COUNT(booking_id) AS total_bookings FROM bookings
GROUP BY flight_id;
```

This query counts the bookings per flight by aggregating the results based on flight_id. The COUNT function aggregates the number of bookings per flight, which helps to know how many reservations each flight has received. It helps to find out the popularity of individual flights fig 20 and Fig 21.

OUTPUT

```
> SELECT flight_id, COUNT(booking_id) AS total_bookings FROM bookings GROUP BY flight_id;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0001, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0001/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 14:27:01,665 Stage-1 map = 0%,  reduce = 0%
2025-03-12 14:27:07,214 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2025-03-12 14:27:14,585 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.7 sec
MapReduce Total cumulative CPU time: 1 seconds 700 msec
Ended Job = job_1741785508473_0001
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 1.7 sec    HDFS Read: 2396 HDFS Write: 300 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 700 msec
OK
101      2
102      2
103      2
104      2
105      2
106      2
107      2
108      2
109      2
110      2
111      1
112      1
113      1
114      1
115      1
116      1
117      1
118      1
119      1
120      1
121      1
122      1
123      1
124      1
```

Figure 20 Screenshot of Total number of bookings per flight part-1

```
116      1
117      1
118      1
119      1
120      1
121      1
122      1
123      1
124      1
125      1
126      1
127      1
128      1
129      1
130      1
131      1
132      1
133      1
134      1
135      1
136      1
137      1
138      1
139      1
140      1
141      1
142      1
143      1
144      1
145      1
146      1
147      1
148      1
149      1
150      1
Time taken: 23.473 seconds, Fetched: 50 row(s)
hive>
```

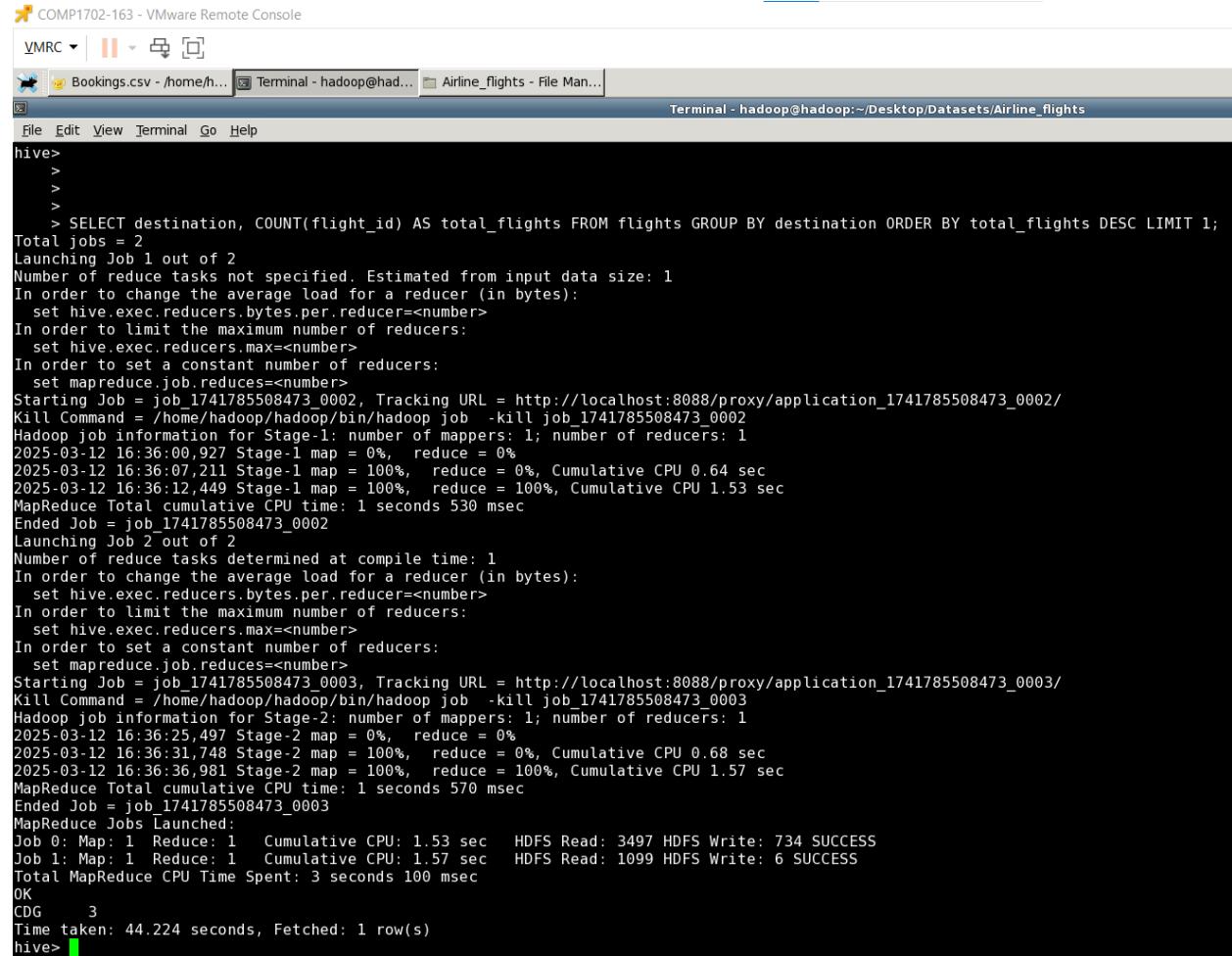
Figure 21 Screenshot of Total number of bookings per flight part-2

ii) The most popular destination

```
SELECT destination, COUNT(flight_id) AS total_flights FROM flights GROUP BY destination ORDER BY total_flights DESC LIMIT 1;
```

This query returns the most flights to a destination. It aggregates the data by destination and sums the number of flights per destination. The result is ordered by the number of flights descending and LIMIT 1 gives only the most visited destination Fig 22.

OUTPUT



The screenshot shows a VMware Remote Console window with a terminal session titled "Terminal - hadoop@hadoop". The terminal output displays the execution of a Hive query to find the most popular destination. The query is as follows:

```
hive> >
>
> SELECT destination, COUNT(flight_id) AS total_flights FROM flights GROUP BY destination ORDER BY total_flights DESC LIMIT 1;
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0002, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0002/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 16:36:00,927 Stage-1 map = 0%,  reduce = 0%
2025-03-12 16:36:07,211 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.64 sec
2025-03-12 16:36:12,449 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.53 sec
MapReduce Total cumulative CPU time: 1 seconds 530 msec
Ended Job = job_1741785508473_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0003, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0003/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0003
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-03-12 16:36:25,497 Stage-2 map = 0%,  reduce = 0%
2025-03-12 16:36:31,748 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.68 sec
2025-03-12 16:36:36,981 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 1.57 sec
MapReduce Total cumulative CPU time: 1 seconds 570 msec
Ended Job = job_1741785508473_0003
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1  Cumulative CPU: 1.53 sec  HDFS Read: 3497 HDFS Write: 734 SUCCESS
Job 1: Map: 1  Reduce: 1  Cumulative CPU: 1.57 sec  HDFS Read: 1099 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 100 msec
OK
CDG      3
Time taken: 44.224 seconds, Fetched: 1 row(s)
hive> 
```

Figure 22 Screenshot of the most popular destination.

iii) The most expensive ticket booked

```
SELECT * FROM bookings
ORDER BY price DESC
```

This query retrieves the booking with the highest price. By ordering the bookings table by price in descending order, the query highlights the most expensive ticket booked. The LIMIT 1 ensures that only the top record is returned Fig 23.

OUTPUT

```

hive>
>
>
> SELECT * FROM bookings
> ORDER BY price DESC
> LIMIT 1;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0005, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0005/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 17:09:45,568 Stage-1 map = 0%, reduce = 0%
2025-03-12 17:09:50,787 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.68 sec
2025-03-12 17:09:57,009 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 1.61 sec
MapReduce Total cumulative CPU time: 1 seconds 610 msec
Ended Job = job_1741785508473_0005
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1   Cumulative CPU: 1.61 sec   HDFS Read: 2396 HDFS Write: 38 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 610 msec
OK
1034    34     134    2024-04-03      Business        620.0
Time taken: 19.17 seconds, Fetched: 1 row(s)
hive>
>
```

Figure 23 Screenshot of the most expensive ticket booked

IV) The average price of Economy-class tickets

```

SELECT AVG(price) AS avg_economy_price
FROM bookings
WHERE seat_class = 'Economy';
```

This query calculates the average ticket price for passengers who booked an Economy seat. The AVG function computes the mean value of the price column for records where seat_class is 'Economy'. This query helps analyze pricing trends within the economy class Fig 24.

OUTPUT

```
hive>
>
>
>
>   > SELECT AVG(price) AS avg_economy_price
>   > FROM bookings
>   > WHERE seat_class = 'Economy';
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job job_1741785508473_0006, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0006/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0006
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 17:15:04,170 Stage-1 map = 0%, reduce = 0%
2025-03-12 17:15:10,409 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.16 sec
2025-03-12 17:15:16,622 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.1 sec
MapReduce Total cumulative CPU time: 2 seconds 100 msec
Ended Job = job_1741785508473_0006
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1   Cumulative CPU: 2.1 sec   HDFS Read: 2396 HDFS Write: 18 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 100 msec
OK
301.111111111111
Time taken: 20.15 seconds, Fetched: 1 row(s)
hive>
```

Figure 24 Screenshot of the average price of Economy-class tickets

- v) Passengers list booked flights from USA

```
SELECT p.name, p.country, b.booking_id, f.origin, f.destination
FROM passengers p
JOIN bookings b ON p.passenger_id = b.passenger_id
JOIN flights f ON b.flight_id = f.flight_id
WHERE p.country = 'USA';
```

This query identifies all passengers from the USA who have booked flights. It joins the passengers, bookings, and flights tables, combining passenger information with booking and flight details. The WHERE clause filters the records to include only passengers from the USA.

OUTPUT

```
COMP1702-163 - VMware Remote Console

VMRC | Booksings.csv - /home/had... | Terminal - hadoop@had... | Airline_flights - File Man...
Terminal - hadoop@hadoop:~$ Terminal - hadoop@hadoop:~$ Desktop/Datasets/Airline_flights

File Edit View Terminal Go Help
> SELECT p.name, p.country, b.booking_id, f.origin, f.destination FROM passengers p JOIN bookings b ON p.passenger_id = b.passenger_id JOIN flights f ON b.flight_id = f.flight_id
> WHERE p.country = 'USA';
Total jobs = 1
2025/03/12 17:24:42 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2025/03/12 17:24:42 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10007/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval; ignoring.
2025/03/12 17:24:42 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10007/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.attempts; ignoring.
Execution Log at: /tmp/hadoop/hadoop_20250312172424.dcbf03c5-c514-467d-a0d7-b378e8011lc.log
2025-03-12 05:24:43 Starting to launch local task to process map join: maximum memory = 518979584
2025-03-12 05:24:44 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10004/HashTable-Stage-5/MapJoin-mapfile31.. hashtable
2025-03-12 05:24:44 Uploaded 1 File to: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10004/HashTable-Stage-5/MapJoin-mapfile31.. hashtable (1671 bytes)
2025-03-12 05:24:44 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10004/HashTable-Stage-5/MapJoin-mapfile21.. hashtable
2025-03-12 05:24:44 Uploaded 1 File to: file:/tmp/hadoop/hive_2025-03-12_17-24-40_154_1775886080469251562-1-local-10004/HashTable-Stage-5/MapJoin-mapfile21.. hashtable (1641 bytes)
2025-03-12 05:24:44 End of local task; Time Taken: 0.708 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1741785508473_0008, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0008/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0008
Hadoop job information for Stage-0: number of mappers: 1; number of reducers: 0
2025-03-12 17:24:50.773 Stage-0 map = 0%, reduce = 0%
2025-03-12 17:24:50.773 Stage-0 map = 100%, reduce = 0%, Cumulative CPU 1.3 sec
MapReduce Total cumulative CPU time: 1 seconds 300 msec
Ended Job = job_1741785508473_0008
MapReduce Jobs Launched:
Job 0: Map: 1 Cumulative CPU: 1.3 sec HDFS Read: 3210 HDFS Write: 370 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 300 msec
OK
Emily Davis USA 1004 LHR JFK
Emily Smith USA 1031 DME SVO
Olivia Johnson USA 1032 DEL BLR
William Brown USA 1033 AKL SYD
Emily Davis USA 1034 DXB LHR
James Miller USA 1035 AMS BCN
Isabella Wilson USA 1036 ZRH LAX
Michael Taylor USA 1037 DEL BLR
Emma Anderson USA 1038 SVO LED
David Thomas USA 1039 IST SGN
Charlotte Moore USA 1040 FRA CDG
Jessica Taylor USA 1056 DXB SYD
Time taken: 17.929 seconds, Fetched: 12 row(s)
hive>
>
```

Figure 25 Screenshot of passengers list booked flights from USA

vi) Flights details departing on a specific date (2024-04-01):

```
SELECT * FROM flights  
WHERE departure_time >= '2024-04-01 00:00:00'  
AND departure_time <= '2024-04-01 23:59:59';
```

This query filters flights based on a specific departure date (April 1, 2024). The WHERE clause restricts the departure_time to be within the specified date range, ensuring that only flights departing on that day are selected Fig 26.

OUTPUT

```
Time taken: 17.922 seconds, Fetched: 12 row(s)
hive>
>
>
>
>
> SELECT * FROM flights
> WHERE departure_time >= '2024-04-01 00:00:00'
> AND departure_time <= '2024-04-01 23:59:59';
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1741785508473_0009, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0009/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0009
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2025-03-12 17:28:25,740 Stage-1 map = 0%, reduce = 0%
2025-03-12 17:28:31,958 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.19 sec
MapReduce Total cumulative CPU time: 1 seconds 190 msec
Ended Job = job_1741785508473_0009
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.19 sec   HDFS Read: 3497 HDFS Write: 67 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 190 msec
OK
123    China Airlines    TPE    HKG    2024-04-01 01:30:00    2024-04-01 05:30:00
Time taken: 12.91 seconds, Fetched: 1 row(s)
hive>
>
```

Figure 26 Screenshot of flights details departing on a specific date (2024-04-01)

vii) The busiest flight route (most booked)

```
SELECT f.origin, f.destination, COUNT(b.booking_id) AS total_bookings
FROM flights f
JOIN bookings b ON f.flight_id = b.flight_id
GROUP BY f.origin, f.destination
ORDER BY total_bookings DESC
LIMIT 1;
```

This query identifies the busiest flight route by counting the number of bookings for each origin-destination pair. It joins the flights and bookings tables on flight_id, groups the data by origin and destination, and orders it by total_bookings in descending order. The result is limited to the busiest route Fig 27 and Fig 28.

OUTPUT

```

COMP1702-163 - VMware Remote Console
VMRC | ||| Terminal - hadoop@had... Airline_flights - File Man...
Bookings.csv - /home/h... Terminal - hadoop@had... Airline_flights - File Man...
Terminal - hadoop@hadoop:~/Desktop/Datasets/Airline_flights
File Edit View Terminal Go Help
> SELECT f.origin, f.destination, COUNT(b.booking_id) AS total_bookings
> FROM flights f
> JOIN bookings b ON f.flight_id = b.flight_id
> GROUP BY f.origin, f.destination
> ORDER BY total_bookings DESC
> LIMIT 1;
Total jobs = 2
2025-03-12 17:41:08 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform.. using builtin-java classes where applicable
2025-03-12 17:41:08 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10008/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval; Ignoring.
2025-03-12 17:41:08 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10008/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval; Ignoring.
Execution log at: /tmp/hadoop/hadoop_20250312174141_3457ff2f-04fa-4f57-8c9a-bf91b13ea5f.log
2025-03-12 05:41:09 Starting to launch local task to process map join; maximum memory = 518979584
2025-03-12 05:41:09 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile41-.hashtable
2025-03-12 05:41:09 Uploaded 1 File to: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile41-.hashtable (1468 bytes)
2025-03-12 05:41:10 End of local task; Time Taken: 0.669 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job 1: job_1741785508473_0010
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 17:41:16:118 Stage-2 map = 0%, reduce = 0%
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.03 sec
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.71 sec
MapReduce Total cumulative CPU time: 1 seconds 710 msec
Ended Job = job_1741785508473_0010
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job 2: job_1741785508473_0011
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0011
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-03-12 17:41:16:118 Stage-2 map = 0%, reduce = 0%
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.83 sec
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.71 sec
MapReduce Total cumulative CPU time: 1 seconds 710 msec
Ended Job = job_1741785508473_0011
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job 2: job_1741785508473_0011
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0011
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1
2025-03-12 17:41:40:116 Stage-3 map = 0%, reduce = 0%
2025-03-12 17:41:45:292 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.65 sec
2025-03-12 17:41:51:481 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.54 sec
MapReduce Total cumulative CPU time: 1 seconds 540 msec
Ended Job = job_1741785508473_0011
Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 1.71 sec HDFS Read: 3497 HDFS Write: 1318 SUCCESS
Job 1: Map: Reduce: 1 Cumulative CPU: 1.54 sec HDFS Read: 1683 HDFS Write: 10 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 250 msec
OK
LHR JFK 3
Time taken: 46.029 seconds, Fetched: 1 row(s)
hive>
```

Figure 27 Screenshot of the busiest flight route (most booked) part -1

```

COMP1702-163 - VMware Remote Console
VMRC | ||| Terminal - hadoop@had... Airline_flights - File Man...
Bookings.csv - /home/h... Terminal - hadoop@had... Airline_flights - File Man...
Terminal - hadoop@hadoop:~/Desktop/Datasets/Airline_flights
File Edit View Terminal Go Help
> SELECT b.booking_id, p.name, f.airline, f.origin, f.destination, b.seat_class, b.price
> FROM bookings b
> JOIN passengers p ON b.passenger_id = p.passenger_id
> JOIN flights f ON b.flight_id = f.flight_id;
Total jobs = 2
2025-03-12 05:41:09 Starting to launch local task to process map join; maximum memory = 518979584
2025-03-12 05:41:09 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile41-.hashtable
2025-03-12 05:41:09 Uploaded 1 File to: file:/tmp/hadoop/hive_2025-03-12_17-41-06_551_2121403765912654973-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile41-.hashtable (1468 bytes)
2025-03-12 05:41:10 End of local task; Time Taken: 0.669 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job 1: job_1741785508473_0010
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-03-12 17:41:16:118 Stage-2 map = 0%, reduce = 0%
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.03 sec
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.71 sec
MapReduce Total cumulative CPU time: 1 seconds 710 msec
Ended Job = job_1741785508473_0010
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job 2: job_1741785508473_0011
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0011
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-03-12 17:41:16:118 Stage-2 map = 0%, reduce = 0%
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.83 sec
2025-03-12 17:41:27:514 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.71 sec
MapReduce Total cumulative CPU time: 1 seconds 710 msec
Ended Job = job_1741785508473_0011
Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 1.71 sec HDFS Read: 3497 HDFS Write: 1318 SUCCESS
Job 1: Map: Reduce: 1 Cumulative CPU: 1.54 sec HDFS Read: 1683 HDFS Write: 10 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 250 msec
OK
LHR JFK 3
Time taken: 46.029 seconds, Fetched: 1 row(s)
hive>
```

Figure 28 Screenshot of the busiest flight route (most booked) part-2

viii) Detailed Booking Information

```

SELECT b.booking_id, p.name, f.airline, f.origin, f.destination, b.seat_class, b.price
FROM bookings b
JOIN passengers p ON b.passenger_id = p.passenger_id
JOIN flights f ON b.flight_id = f.flight_id;

```

This query retrieves detailed booking information by joining the bookings, passengers, and flights tables. It returns details such as booking_id, passenger name, airline, origin, destination, seat class, and price for each booking, providing a comprehensive view of each booking transaction Fig 29 and Fig 30.

OUTPUT

```

COMP1702-163 - VMware Remote Console
VMRC | Terminal - hadoop@had... | Airline_flights - File Man...
File Edit View Terminal Go Help
> SELECT b.booking_id, p.name, f.airline, f.origin, f.destination, b.seat_class, b.price
  FROM bookings b
    JOIN passengers p ON b.passenger_id = p.passenger_id
  JOIN flights f ON b.flight_id = f.flight_id;
Total jobs = 1
25/03/12 18:41:31 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/03/12 18:41:31 WARN Configuration: Configuration file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10007/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval: Ignoring
25/03/12 18:41:32 WARN Configuration: Configuration file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10007/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.attempts: Ignoring
25/03/12 18:41:31 Starting to launch local task to process map join; maximum memory = 518979584
2025-03-12 06:41:32 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10004/HashTable-Stage-5/MapJoin-mapfile60--.hashtable
2025-03-12 06:41:32 Uploaded 1 File(s) to: file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10004/HashTable-Stage-5/MapJoin-mapfile60--.hashtable (2660 bytes)
2025-03-12 06:41:32 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10004/HashTable-Stage-5/MapJoin-mapfile51--.hashtable
2025-03-12 06:41:32 Uploaded 1 File(s) to: file:/tmp/hadoop/hive_2025-03-12_18-41-28_728_7891568118156062740-1-local-10004/HashTable-Stage-5/MapJoin-mapfile51--.hashtable (2258 bytes)
2025-03-12 06:41:32 End of local task; Time Taken: 0.901 sec
Execution completed successfully
MapredLocalTask submitted
LocalJob: Job-0001 of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1741785508473_0012, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0012/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0012
Hadoop version: 2.7.3
Map: 100% Reduces: 0%
2025-03-12 18:41:38 779 Stage-S map = 100%, reduce = 0%, Cumulative CPU 0.94 sec
MapReduce Total cumulative CPU time: 940 msec
Ended job: job_1741785508473_0012
MapReduce Jobs Launched:
Job 0: Map: 1 Cumulative CPU: 0.94 sec HDFS Read: 3210 HDFS Write: 3230 SUCCESS
Total MapReduce CPU Time Spent: 940 msec
OK
1001 Rahul Sharma American Airlines JFK LAX Economy 250.0
1002 Alice Smith Delta Airlines ORD MIA Business 450.0
1003 Amit Verma United Airlines LAX DFW Economy 300.0
1004 Emily Davis British Airways LHR JFK Economy 350.0
1005 James Wilson Virgin Atlantic YVR YYZ Economy 500.0
1006 Sarah Wilson Emirates DXB SYD Economy 400.0
1007 David Martinez Qatar Airways DOH BKK Business 550.0
1008 Ishita Banerjee Lufthansa FRA SFO Economy 320.0
1009 James Miller United Airlines SIN NRT Economy 280.0
1010 Sameer Kapoor Turkish Airlines IST JNB Business 600.0
1011 Lucas Müller KLM AMS CDG Economy 230.0
1012 Emma Dubois Air France CDG AMS Business 550.0
1013 Alessandro Romano Swiss Air ZRH LHR Economy 370.0
1014 Sofia Jensen Qantas SYD MEL Economy 260.0

```

Figure 29 Screenshot of detailed Booking Information part-1

```

COMP1702-163 - VMware Remote Console
VMRC | Terminal - hadoop@had... | Airline_flights - File Man...
File Edit View Terminal Go Help
1017 David Eriksson Air India DEL MAA Business 530.0
1018 Isabelle Laurent EasyJet LGW BCN Economy 300.0
1019 Nikola Kovács Flybe EXT MAN Economy 310.0
1020 Camilla Rossi Ryanair DUB STN Business 600.0
1021 James Wilson Jetstar Airways MEL BNE Economy 250.0
1022 Charlotte Brown Alitalia FCO LIN Business 450.0
1023 Oliver Davies China Airlines TPE HKG Economy 350.0
1024 Amelia Johnson Emirates DXB BOM Economy 280.0
1025 William Taylor Finnair HEL STO Business 550.0
1026 Sophia Evans Turkish Airlines IST CAI Economy 400.0
1027 Henry Thomas Virgin Atlantic LHR JFK Business 600.0
1028 Emily Robinson United Airlines EWR SFO Economy 300.0
1029 Jack Harris Lufthansa FRA ORD Business 520.0
1030 Isla Walker American Airlines DFW ORD Economy 280.0
1031 John Smith Qatar Airways DOH SYD Business 540.0
1032 Olivia Johnson Jet Airways DEL BLR Economy 330.0
1033 William Brown Air New Zealand AKL SYD Economy 320.0
1034 Emily Davis Emirates DXB LHR Business 620.0
1035 James Miller KLM AMS BCN Economy 280.0
1036 Isabella Wilson Swiss Air ZRH LAX Business 550.0
1037 Michael Taylor Air India DEL BLR Economy 300.0
1038 Emma Anderson Aeroflot SVO LED Economy 250.0
1039 David Thomas Turkish Airlines IST SGN Business 530.0
1040 Charlotte Moore Lufthansa FRA CDG Economy 310.0
1041 Olga Ivanova United Airlines SFO ORD Economy 330.0
1042 Lucia Garcia Delta Airlines ATL LAX Business 560.0
1043 Aya Nakamura Aeroflot LED HEL Economy 270.0
1044 Chloe Lefevre Jet Airways BOM BLR Business 500.0
1045 Kofi Owusu Virgin Atlantic LHR SFO Economy 280.0
1046 Ahmed Ali Alitalia FCO CDG Business 550.0
1047 Leila Bouzid China Southern Airlines CAN BKK Economy 320.0
1048 Daniela Rodriguez Jetstar Airways BNE MEL Economy 330.0
1049 Martina Novak Finnair HEL AMS Business 620.0
1050 Felipe Silva Turkish Airlines IST CAI Economy 270.0
1051 Amir Khan American Airlines JFK LAX Business 570.0
1052 Claire O'Connor Delta Airlines ORD MIA Economy 240.0
1053 Sarah Ibrahim United Airlines LAX DFW Economy 290.0
1054 Rafael Perez British Airways LHR JFK Business 600.0
1055 Yassir Ben Ali Air Canada YVR YYZ Economy 310.0
1056 Jessica Taylor Emirates DXB SYD Business 550.0
1057 Leandro Silva Qatar Airways DOH BKK Economy 250.0
1058 Marco González Lufthansa FRA SFO Business 530.0
1059 Maria Kovács Singapore Airlines SIN NRT Economy 300.0
1060 Johann Schmidt Turkish Airlines IST JNB Economy 320.0
Time taken: 18.376 seconds, Fetched: 60 row(s)
hive>
```

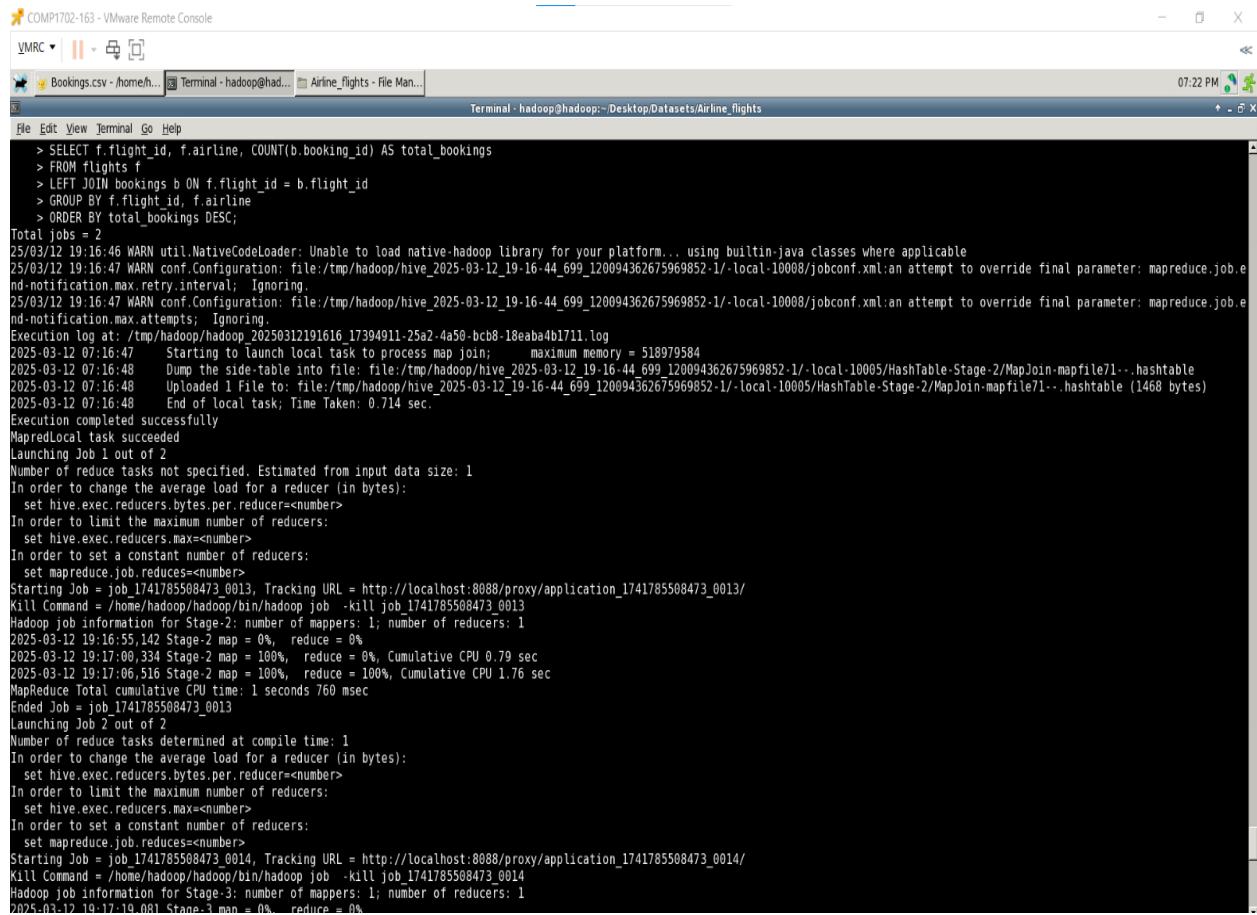
Figure 30 Screenshot of detailed Booking Information part-2

ix)Bookings per Flight

```
SELECT f.flight_id, f.airline, COUNT(b.booking_id) AS total_bookings
FROM flights f
LEFT JOIN bookings b ON f.flight_id = b.flight_id
GROUP BY f.flight_id, f.airline
ORDER BY total_bookings DESC;
```

This query calculates the capacity utilization of each flight by counting the number of bookings for each flight. It uses a LEFT JOIN to ensure all flights are included, even those with no bookings. The query groups by flight_id and airline, providing insight into how many bookings each flight has received, sorted by total_bookings Fig 31 and Fig 32.

OUTPUT



The screenshot shows a VMware Remote Console window with two tabs: 'Terminal - hadoop@hadoop' and 'Airline_flights - File Man...'. The terminal tab displays the execution of a Hive query:

```
> SELECT f.flight_id, f.airline, COUNT(b.booking_id) AS total_bookings
> FROM flights f
> LEFT JOIN bookings b ON f.flight_id = b.flight_id
> GROUP BY f.flight_id, f.airline
> ORDER BY total_bookings DESC;
Total jobs = 2
25/03/12 19:16:46 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/03/12 19:16:47 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_19-16-44_699_120094362675969852-1-local-10008/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval: Ignoring
25/03/12 19:16:47 WARN conf.Configuration: file:/tmp/hadoop/hive_2025-03-12_19-16-44_699_120094362675969852-1-local-10008/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.attempts: Ignoring.
Execution log at: /tmp/hadoop/hadoop_20250312191616_17394911-25a2-4a50-bc8-18eaba4b1711.log
2025-03-12 07:16:47 Starting to launch local task to process map join; maximum memory = 518979584
2025-03-12 07:16:48 Dump the side-table into file: file:/tmp/hadoop/hive_2025-03-12_19-16-44_699_120094362675969852-1-local-10005/HashTable-Stage-2/MapJoin-mapfile71--.hashtable
2025-03-12 07:16:48 Uploaded 1 File to: file:/tmp/hadoop/hive_2025-03-12_19-16-44_699_120094362675969852-1-local-10005/HashTable-Stage-2/MapJoin-mapfile71--.hashtable (1468 bytes)
2025-03-12 07:16:48 End of local task; Time Taken: 0.714 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0013, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0013/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0013
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2025-03-12 19:16:55.142 Stage-2 map = 0%, reduce = 0%
2025-03-12 19:17:08.334 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.79 sec
2025-03-12 19:17:06.516 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.76 sec
MapReduce Total cumulative CPU time: 1 seconds 760 msec
Ended Job = job_1741785508473_0013
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1741785508473_0014, Tracking URL = http://localhost:8088/proxy/application_1741785508473_0014/
Kill Command = /home/hadoop/hadoop/bin/hadoop job -kill job_1741785508473_0014
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1
2025-03-12 19:17:19.081 Stage-3 map = 0%, reduce = 0%
```

Figure 31 Screenshot of bookings per Flight part-1

```

File Edit View Terminal Go Help
104 British Airways 2
103 United Airlines 2
102 Delta Air Lines 2
101 American Airlines 2
150 Turkish Airlines 1
149 Finnair 1
148 Jetstar Airways 1
147 China Southern Airlines 1
146 Alitalia 1
145 Virgin Atlantic 1
144 Jet Airways 1
143 Qantas 1
142 Delta Airlines 1
141 United Airlines 1
140 Lufthansa 1
139 Turkish Airlines 1
138 Aeroflot 1
137 Air India 1
136 Swiss Air 1
135 KLM 1
134 Emirates 1
133 Air New Zealand 1
132 Jet Airways 1
131 Qatar Airways 1
130 American Airlines 1
129 Lufthansa 1
128 United Airlines 1
127 Virgin Atlantic 1
126 Turkish Airlines 1
124 Emirates 1
123 China Airlines 1
122 Alitalia 1
121 Jetstar Airways 1
120 Ryanair 1
119 Flybe 1
118 Easyjet 1
117 Air India 1
116 Jet Airways 1
115 Aeroflot 1
114 Qantas 1
113 Swiss Air 1
112 Air France 1
111 KLM 1
125 Finnair 1
Time taken: 46.853 seconds, Fetched: 50 row(s)
hive>

```

Figure 32 Screenshot of bookings per Flight part-2

x) The Most Frequent Routes (Origin → Destination)

```

SELECT f.origin, f.destination, COUNT(b.booking_id) AS total_flights
FROM flights f
JOIN bookings b ON f.flight_id = b.flight_id
GROUP BY f.origin, f.destination
ORDER BY total_flights DESC
LIMIT 5;

```

This query identifies the most frequent routes by counting the total number of flights booked on each route. It groups the data by origin and destination and orders the results by total_flights in descending order, returning the top 5 most frequent routes. This helps the airline analyze which routes are most popular with passengers Fig 33.

```

2020-08-12 19:17:23,808 Stage-3 Map: 100%, Reduced: 100%, Cumulative CPU: 1700 msec
MapReduce Total cumulative CPU time: 1 seconds 530 msec
Ended Job = job_1741785508473_0017
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Cumulative CPU: 1.74 sec HDFS Read: 3497 HDFS Write: 1318 SUCCESS
Job 1: Map: 1 Reduce: 1 Cumulative CPU: 1.53 sec HDFS Read: 1683 HDFS Write: 50 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 270 msec
OK
LHR      JFK      3
JFK      LAX      2
YVR      YYZ      2
DEL      BLR      2
IST      JNB      2
Time taken: 46.006 seconds, Fetched: 5 row(s)
hive>

```

Figure 33 Screenshot of the Most Frequent Routes (Origin → Destination)

These queries provided valuable insights for airline operations, including finding out popular routes, understanding passenger behavior, and setting optimal pricing. The utilization of Hive in Hadoop VM was successful in handling structured airline data and enabling efficient data-driven decision-making.

3. Task C - MapReduce Programming

Student Information

Student ID: 001354599 (Last digit = 9)

Selected Task: Task 5 - Output the number of courses for each year

3.1 Problem Definition

Given a large dataset of student records in the below format:

(Student_Id, Age, Course_Id, Course_Grade, School, Year)

Example Data

SC001238, 25, COMP1702, 65, CMS school, 2019

SM100235, 21, ECON1011, 77, BS school, 2017

Objective is to aim the use of MapReduce framework to determine the number of courses for each year in an efficient and scalable manner.

3.2 MapReduce Algorithm Design

3.2.1 Mapping Stage

Input Data:

Each line of the dataset contains the following information:

(Student_Id, Age, Course_Id, Course_Grade, School, Year)

For example:

SC001238, 25, COMP1702, 65, CMS school, 2019

SM100235, 21, ECON1011, 77, BS school, 2017

Key-Value Pair Mapping:

The mapper generates a key-value pair and extracts the Year field from each record during the map stage. The Year is the key, and one course for that year is represented by the value 1.

Key: In the MapReduce process, the Year—which is taken from the record's sixth field—acts as the key. In the dataset, the key will stand for a specific year.

Value: Each key has a value of 1. The frequency of a course in that year is indicated by this value. To determine how many courses are linked to each year, we generate this value.

Key and Value Computation:

The Year (6th field) is extracted for every input record and used as the key. Because each record relates to a single course for a specific year, we give each record a value of 1. This makes it possible to easily add up the values to determine the total number of courses for each year once the records are sent to the reducer.

3.2.2 Mapper Pseudocode:

```
class Mapper:  
    def initialize():  
        year_counts = {}  
  
    def map(record):  
        fields = record.split(",") # Split the record by commas  
        year = fields[5].strip() # Extract the Year (6th field)  
        if year in year_counts:  
            year_counts[year] += 1 # Increment count for that year  
        else:  
            year_counts[year] = 1 # Initialize count for new year  
  
    def close():  
        # Emit the aggregated counts for each year  
        for year, count in year_counts.items():  
            Emit(year, count)
```

The **mapper** will output the following key-value pairs:

(2019, 1)

(2017, 1)

These key-value pairs indicate that:

- In the year 2019, there is 1 course.
- In the year 2017, there is 1 course.

3.2.4 Combiner Stage

In MapReduce, a combiner acts as a mini-reducer that processes intermediate data from the mapper before it reaches the main reducer. This step plays a crucial role in reducing the amount of data transferred across the network, improving efficiency and scalability, particularly when working with large datasets. By performing local aggregation, the combiner sends fewer, pre-aggregated key-value pairs to the reducer, which helps speed up the entire process.

3.2.5 Combiner Pseudocode:

```
class Combiner:  
    def initialize():  
        combined_counts = {}  
  
    def combine(key, values):  
        total_count = sum(values)      # Aggregate values for each key (year)  
        Emit(key, total_count)         # Emit aggregated result for the key
```

The combiner is especially effective in tasks like counting, where operations are associative and commutative. In this case, it aggregates course counts by year locally, minimizing the amount of intermediate data that needs to be shuffled. This not only reduces network traffic but also speeds up the overall computation, making it a valuable optimization in handling large-scale datasets, like student course data.

Example Output

```
(2019, 1)  
(2017, 1)
```

3.2.6 Stage of Shuffling and Sorting

All key-value pairs are automatically grouped by the key (in this case, the Year) once Hadoop has finished the map stage.

Prior to the reduce stage, the intermediate key-value pairs are sorted by the key (Year). For instance, following the stage of shuffle and sort

```
(2017, [1])  
(2019, [1, 1])
```

3.2.7 Reducing Stage

Input to Reducer:

The key-value pairs are grouped and sent to the reducer:

(2017, [1])

(2019, [1, 1])

Processing the Data:

The reducer adds up the values for every key to determine the total number of courses for each year.

3.2.8 Reducer Pseudocode:

```
class Reducer:  
    def initialize():  
        final_counts = {}  
  
    def reduce(year, values):  
        total_courses = sum(values) # Sum the list of 1s  
        Emit(year, total_courses) # Emit the total count of courses for the year
```

Example Reducer Output:

For (2017, [1]), the total is 1.

For (2019, [1, 1]), the total is 2.

Thus, the final output will be:

(2017, 1)

(2019, 2)

This output indicates:

In the year 2017, there was 1 course.

In the year 2019, there were 2 courses.

3.3 Explanation of the Output

Details of Key-Value Output:

The year (2017, 2019, etc.) is the key.

Value: The number of classes offered annually.

Mapper:

For every record, the mapper generates the pairs (Year, 1).

The combiner output remains the same as the mapper output in this case because each year appears only once in the given input. Since there's nothing to aggregate locally, the combiner simply forwards the counts as they are. However, in a larger dataset with multiple courses per year, the combiner would sum up local counts before sending them to the reducer, reducing network traffic and improving efficiency.

The mapper emits (2019, 1) from the record SC001238, 25, COMP1702, 65, CMS school, 2019.

The mapper generates (2017, 1) from the record SM100235, 21, ECON1011, 77, BS school, 2017.

Shuffling & Sorting: The output is sorted and grouped by year using the MapReduce framework. Groups such as (2017, [1]) and (2019, [1, 1]) are the outcome of this.

Reducer Aggregates: Each year's values are added up by the reducer:

In 2017, the total is 1.

In 2019, the total is 2.

Final Output: The final output consists of the total number of courses per year:

(2017, 1)

(2019, 2)

3.4 Performance Analysis and Algorithm Efficiency

Handling large datasets efficiently is a crucial challenge in big data processing, particularly when working with structured records like student enrollments. This analysis explores how a MapReduce algorithm is designed to count the number of courses per year while ensuring optimal memory usage, improved performance, scalability, and fault tolerance.

One of the key strengths of the algorithm is its ability to reduce data transfer between nodes by using local aggregation techniques such as combiners and in-mapper combining. These methods allow data to be processed locally before being sent to the reducer, minimizing network congestion and improving overall execution speed. Additionally, the shuffling and sorting phase of MapReduce ensures that data is grouped by year before reaching the reducer. This built-in

organization significantly reduces computational overhead, allowing the reducer to process data more efficiently.

Another major advantage is efficient memory utilization. Instead of repeatedly writing intermediate results to disk, the algorithm stores counts in memory and emits the final results only at the end of the mapping phase. This approach minimizes disk I/O operations, which can otherwise slow down processing, leading to faster execution times and improved system performance.

Scalability is a fundamental aspect of this solution. By distributing computations across multiple machines, the MapReduce framework ensures that the algorithm can handle large-scale datasets efficiently. The use of parallel processing helps distribute the workload evenly, preventing system bottlenecks and maintaining consistent performance, even as the dataset grows in size.

To further enhance efficiency, the algorithm is designed to reduce unnecessary data movement. A combiner function is used to aggregate partial results locally before passing them to the reducer. This significantly reduces the volume of data being transferred, optimizing network bandwidth usage and accelerating the overall processing time. Furthermore, in-memory aggregation helps minimize reliance on disk storage, leading to even greater performance improvements.

Reliability is another major advantage. Thanks to the fault tolerance mechanisms in Hadoop, any failed tasks are automatically reassigned to other nodes, ensuring that processing continues without disruption. This built-in redundancy prevents data loss and enhances the system's overall stability, making the algorithm a robust and dependable solution for large-scale data processing.

In conclusion, this MapReduce solution successfully balances efficiency, scalability, and fault tolerance. By leveraging parallel processing, memory optimization, and reduced network overhead, it provides a highly effective method for aggregating large datasets. The use of combiners and in-mapper aggregation ensures minimal data movement, while Hadoop's fault tolerance capabilities enhance reliability. These combined factors make this algorithm an ideal approach for processing massive datasets in distributed computing environments.

4. Task D Big Data Project Analysis – PA Company

Introduction

The PA company, a leading provider of precision agriculture services, is embarking on a big data project to revolutionize farming and agricultural decision-making. By leveraging data-driven insights, the company aims to help farmers and agribusinesses worldwide in understanding weather implications, optimizing resource allocation, improving crop yields, and mitigating risks. To achieve this, vast amounts of data—ranging from satellite images and IoT sensor feeds to market trends and social media analytics—must be efficiently processed and analyzed. Given the complexity and scale of this initiative, critical decisions must be made regarding data storage, real-time analytics, and cloud infrastructure.

This report examines the most suitable data storage architecture, evaluates real-time processing frameworks, and proposes a secure and scalable cloud-hosting strategy to ensure the success of this big data initiative.

4.1 Task D.1: Data Storage – Data Warehouse vs. Data Lake

The PA company plans to store over 300 petabytes of agricultural data, so it must choose between a data warehouse and a data lake. While data warehouses have traditionally been used for business intelligence and structured data storage, data lakes offer a more flexible solution that supports structured, semi-structured, and unstructured data. Given the range of data sources, including social media posts, satellite imagery, historical weather reports, and real-time sensor streams, a data lake is the most practical choice.

A data lake is perfect for handling vast volumes of heterogeneous data because it enables schema-on-read processing, which permits flexible analysis free from the constraints of a predefined schema (Davenport, 2020). This technique enables batch and real-time analytics by allowing the ingestion of raw data.

The table below compares key attributes of data warehouses and data lakes:

Feature	Data Warehouse	Data Lake
Data Type	Structured (SQL, relational data)	Structured, semi-structured, unstructured (IoT, videos, text)
Processing Speed	Optimized for batch queries	Supports real-time and batch processing
Scalability	Limited by schema restrictions	Highly scalable with cloud storage
Flexibility	Requires predefined schema	Adapts to evolving data formats
Cost Efficiency	Expensive due to storage and compute requirements	Cost-effective due to pay-as-you-go models

Table 1: Features of Data Warehouse and Data Lakes.

A data lake architecture allows for seamless integration of diverse data formats, supporting advanced machine learning, predictive analytics, and real-time monitoring—all essential for precision agriculture.

By implementing a cloud-based data lake, PA company can ensure highly scalable, cost-effective, and adaptable data storage, positioning itself for advanced agricultural insights and automation.

4.2 Task D.2: Real-Time Processing – Evaluating MapReduce and Alternatives

Given the nature of precision agriculture, PA company requires real-time data analytics to support instant decision-making on weather patterns, crop health, and resource allocation. Some IT managers have suggested using MapReduce, a well-known distributed computing framework, to process incoming data. However, while MapReduce excels at batch processing and large-scale data transformation, it suffers from high latency, making it unsuitable for real-time applications (Dean & Ghemawat, 2008).

Preferred Alternative: Apache Kafka & Spark Streaming

For real-time analytics, a combination of Apache Kafka and Spark Streaming presents a more effective solution. Kafka enables event-driven data ingestion, while Spark Streaming processes incoming data in micro-batches, ensuring low-latency analytics.

The following table compares different frameworks based on their suitability for real-time analytics:

Framework	Processing Type	Latency	Best Use Case
MapReduce	Batch processing	High (minutes-hours)	Historical analysis
Apache Spark Streaming	Micro-batch processing	Low (seconds)	Sensor and market trend analytics
Apache Flink	True real-time streaming	Very low (milliseconds)	Instant event detection
Apache Kafka + Spark	Event-driven streaming	Low (milliseconds-seconds)	Real-time monitoring

Table 2 Features of Different framework.

Data Processing Pipeline flows from IoT sensors, satellites, and online sources into Kafka, which distributes it to Spark Streaming for instant processing. This ensures real-time insights, allowing farmers and agricultural enterprises to make timely decisions.

By leveraging Kafka and Spark Streaming, PA company can maintain high-speed, scalable, and efficient real-time analytics, significantly improving decision-making accuracy in agriculture.

4.3 Task D.3: Cloud Hosting Strategy – Ensuring Security, Scalability, and Availability

As PA company plans to migrate its applications and services to the cloud, it must prioritize security, scalability, and high availability. Given the sensitive nature of some datasets—such as customer information and financial records—a hybrid cloud approach is recommended. This model ensures sensitive data is stored securely in a private cloud, while public cloud services handle large-scale agricultural data processing (Fernandes et al., 2014).

Proposed Cloud Architecture

Cloud Component	Purpose	Example Services
Compute Layer	Handles processing workloads	AWS EC2, Azure VM, Google Compute Engine
Storage Layer	Manages structured & unstructured data	AWS S3, Azure Data Lake, Google Cloud Storage
Security Layer	Ensures data protection & compliance	AWS IAM, Azure Security Center
Networking Layer	Provides global access & redundancy	AWS CloudFront, Azure CDN

Table 3: Comparison of Cloud Architecture with different components.

A hybrid cloud strategy enables efficient data storage and processing, while ensuring compliance with security regulations and performance optimization.

Security, Scalability, and High Availability

1. Security Measures: Private cloud storage encrypts customer data, implementing role-based access control (RBAC) and multi-factor authentication (MFA) to prevent breaches.
2. Scalability: Auto-scaling features in AWS and Azure allow the system to adapt dynamically to workload fluctuations.
3. High Availability: Load balancing and multi-region deployment ensure uninterrupted service availability, even during system failures.

By adopting this hybrid cloud model, PA company can achieve secure, scalable, and highly available infrastructure, ensuring efficiency and innovation in precision agriculture.

Reference

Dean, J. and Ghemawat, S. (2004) 'MapReduce: Simplified Data Processing on Large Clusters', *Communications of the ACM*, 51(1), pp.107-113. Available at:

<https://dl.acm.org/doi/10.1145/1327452.1327492>

Holmes, A. (2012) *Hadoop in Practice*. 1st ed. Shelter Island, NY: Manning Publications.

Available at:

https://www.google.co.uk/books/edition/Hadoop_in_Practice/tzkzEAAAQBAJ?hl=en&gbpv=1&dq=hadoop+in+practice+by+O%27Neill,+M.&printsec=frontcover

Shvachko, K., Kuang, H., Radia, S. & Chansler, R. (2010) 'The Hadoop Distributed File System', *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, pp. 1-10. Available at:

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=08d12e771d811bcd0d4bc81fa3993563efbaedb>

Davenport, T.H. (2014) *Big Data at Work: Dispelling the Myths, Uncovering the Opportunities*.

1st edn. Boston: Harvard Business Review Press. Available at:

https://books.google.co.uk/books?id=apjBAGAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Dean, J. and Ghemawat, S. (2008) 'MapReduce: Simplified Data Processing on Large Clusters', *Communications of the ACM*, 51(1), pp. 107-113. Available at:

<https://doi.org/10.1145/1327452.1327492>.

Hashizume, K., Rosado, D. G., Fernández-Medina, E. and Fernandez, E. B. (2013) 'An analysis of security issues for cloud computing', *Journal of Internet Services and Applications*, 4(1), pp. 1-13.

Available at: <https://doi.org/10.1186/1869-0238-4-5> (