Program 5

Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB

MongoDB: A Powerful NoSQL Database

MongoDB is a popular open-source, non-relational database management system (DBMS) that stores data in flexible, JSON-like documents. Unlike traditional relational databases (RDBMS) that use tables and rows, MongoDB utilizes collections and documents for data storage and retrieval.

MongoDB's key features:

1. **Document-oriented model**
   - Data is stored in documents similar to JSON objects (actually BSON, a binary JSON format), making it intuitive for developers and aligning well with object-oriented programming.
   - Documents are grouped into collections, similar to tables in RDBMS, according to Board Infinity.
   - Offers a flexible schema, meaning documents within a collection can have different fields, allowing for dynamic changes to the data model without downtime

2. **High scalability**
   - Supports horizontal scaling through sharding, distributing data across multiple servers to handle large datasets and high traffic efficiently.
   - Replica sets provide high availability and redundancy by maintaining multiple copies of data across different servers, enabling automatic failover in case of primary server failure

3. **Performance**
   - Optimized for high read and write throughput, suitable for applications demanding rapid data processing.
   - Leverages indexing, replication, and sharding to enhance performance and manage intensive workloads.
   - Its in-memory storage engine contributes to fast performance, especially for read-heavy operations.

4. **Rich query language and aggregation**
   - Provides a powerful query language (MQL) that's flexible and allows for complex queries, including field, range, and regular expression searches.
   - The aggregation framework enables sophisticated data transformations and aggregations within the database, according to Ksolves.

5. **Other important features**
    - Ad-hoc queries: Supports flexible and real-time queries without predefined schemas.
    - Geospatial data support: Offers built-in capabilities for applications requiring location-based services.
    - Transactions: Supports multi-document ACID transactions, though they might not be as mature or efficient as in traditional RDBMS.
    - Built-in security: Includes authentication mechanisms like SCRAM and role-based access controls

Advantages of MongoDB

- **Scalability**: Handles large datasets and traffic spikes effectively via horizontal scaling.
- **Flexibility**: Adapts easily to evolving data requirements due to its schema-less nature.
- **Performance**: Delivers high read and write performance, particularly for large volumes of data.
- **Developer-friendly**: Intuitive document model and query language simplify development and reduce the need for complex object-relational mapping (ORM).
- **Cloud-native**: Offers MongoDB Atlas, a fully managed cloud database service on major cloud providers like AWS, Azure, and Google Cloud, says MongoDB

Disadvantages of MongoDB

- **Memory Usage**: Can be memory-intensive, especially for large datasets, potentially leading to higher resource costs.
- **Transactions**: Although MongoDB has transaction capabilities, complex transactions across multiple operations might be less robust compared to RDBMS.
- **Consistency**: MongoDB prioritizes scalability and availability, potentially leading to eventual consistency rather than immediate consistency in certain scenarios.
- **Indexing limitations**: While robust, incorrect or excessive indexing can impact write performance.
- **Data Duplication**: Denormalized data modeling can lead to redundancy and increased storage.

MongoDB Operations: Count, Sort, Limit, Skip, Aggregate

shop_db> use college_db

switched to db college_db

```
test> use college_db
switched to db college_db
college_db> db.student.insertOne
... ({
...   "_id": 8,
...   "name": "Kavin",
...   "age": 24,
...   "marks": 82,
...   "department": "ISE"
... })
...
{ acknowledged: true, insertedId: 8 }
college_db>
```

1. Count Documents

Count all documents:

db.student.countDocuments()

Count with a condition (e.g., marks > 80):

db.student.countDocuments({ marks: { $gt: 80 } })

```
college_db> db.student.countDocuments();
5
college_db> db.student.countDocuments({ marks: { $gt: 80 } })
3
college_db>
```

2. Sort Documents

Sort by marks in descending order:

db.student.find().sort({ marks: -1 })

```
college_db> db.student.find().sort({ marks: -1 })
[
  { _id: 4, name: 'Grace', age: 22, marks: 90, department: 'ISE' },
  { _id: 2, name: 'Clara', age: 21, marks: 87, department: 'ISE' },
  { _id: 1, name: 'Alice', age: 21, marks: 85, department: 'CSE' },
  { _id: 3, name: 'Flavy', age: 22, marks: 80, department: 'CSE' },
  { _id: 5, name: 'Frank', age: 22, marks: 75, department: 'CSE' }
]
college_db>
```

Sort by name ascending and age descending:

db.student.find().sort({ name: 1, age: -1 })

```
college_db> db.student.find().sort({ name: 1, age: -1 })
[
  { _id: 1, name: 'Alice', age: 21, marks: 85, department: 'CSE' },
  { _id: 2, name: 'Clara', age: 21, marks: 87, department: 'ISE' },
  { _id: 3, name: 'Flavy', age: 22, marks: 80, department: 'CSE' },
  { _id: 5, name: 'Frank', age: 22, marks: 75, department: 'CSE' },
  { _id: 4, name: 'Grace', age: 22, marks: 90, department: 'ISE' }
]
college_db>
```

3. Limit Results

Return top 5 students:

db.student.find().limit(5)

```
college_db> db.student.insertOne
... ({
...   "_id": 6,
...   "name": "Frank",
...   "age": 24,
...   "marks": 65,
...   "department": "CSE"
... })
...
{ acknowledged: true, insertedId: 6 }
college_db> db.student.find().limit(5)
[
  { _id: 1, name: 'Alice', age: 21, marks: 85, department: 'CSE' },
  { _id: 2, name: 'Clara', age: 21, marks: 87, department: 'ISE' },
  { _id: 3, name: 'Flavy', age: 22, marks: 80, department: 'CSE' },
  { _id: 4, name: 'Grace', age: 22, marks: 90, department: 'ISE' },
  { _id: 5, name: 'Frank', age: 22, marks: 75, department: 'CSE' }
]
college_db>
```

4. Skip Documents

Skip first 5 documents and get the next 5:

db.student.find().skip(5).limit(5)

```
college_db> db.student.find().skip(5).limit(5)
[
  { _id: 6, name: 'Frank', age: 24, marks: 65, department: 'CSE' },
  { _id: 7, name: 'Rosy', age: 24, marks: 88, department: 'ECE' }
]
college_db>
```

5. Aggregate Documents

Group by Department and Count Students:

```
db.student.aggregate([ { $group: { _id: "$department", totalStudents: { $sum: 1 } } } ])
```

```
college_db> db.student.aggregate([ { $group: { _id: "$department", totalStudents: { $sum: 1 } } } ])
[
  { _id: 'ISE', totalStudents: 2 },
  { _id: 'CSE', totalStudents: 4 },
  { _id: 'ECE', totalStudents: 1 }
]
college_db>
```

Group by Department and Average Marks:

```
db.student.aggregate([ { $group: { _id: "$department", avgMarks: { $avg: "$marks" } } } ])
```

```
college_db> db.student.aggregate([ { $group: { _id: "$department", avgMarks: { $avg: "$marks" } } } ])
[
  { _id: 'ISE', avgMarks: 88.5 },
  { _id: 'CSE', avgMarks: 76.25 },
  { _id: 'ECE', avgMarks: 88 }
]
college_db>
```

Filter → Group → Sort → Limit (Full Pipeline):

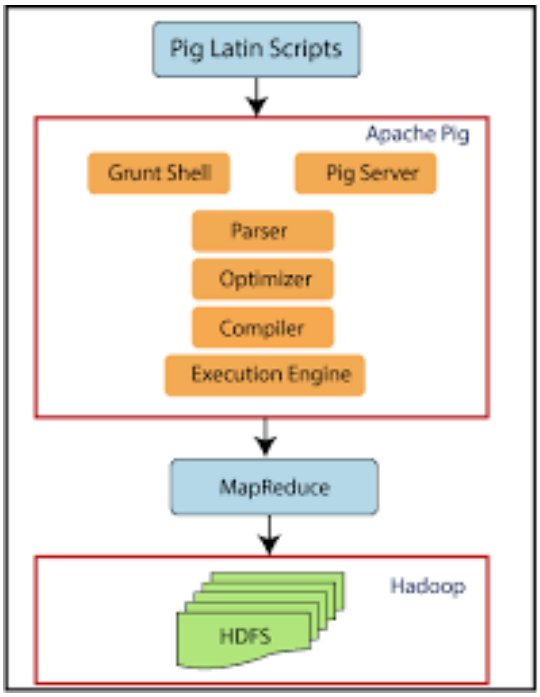
```
db.student.aggregate([
 { $match: { marks: { $gt: 60 } } },
 { $group: { _id: "$department", avgMarks: { $avg: "$marks" } } },
 { $sort: { avgMarks: -1 } },
 { $limit: 3 }
])
```

```
college_db> db.student.aggregate([
...   { $match: { marks: { $gt: 60 } } },
...   { $group: { _id: "$department", avgMarks: { $avg: "$marks" } } },
...   { $sort: { avgMarks: -1 } },
...   { $limit: 3 }
... ])
...
[
  { _id: 'ISE', avgMarks: 88.5 },
  { _id: 'ECE', avgMarks: 88 },
  { _id: 'CSE', avgMarks: 76.25 }
]
college_db>
```

Program 6

Write Pig Latin scripts to sort, group, join, project, and filter the data.

Pig Latin scripts are used with Apache Pig, a high-level platform for processing large datasets, to create data analysis codes. These scripts are written in Pig Latin, a language that abstracts the complexities of MapReduce, allowing users to focus on data analysis rather than low-level programming. Pig Latin scripts are submitted to the Apache Pig server, parsed, optimized, and then compiled into MapReduce code for execution on a Hadoop cluster.



Breakdown of key aspects:

1. Purpose:

- Pig Latin scripts are used to analyze data stored in Hadoop's distributed file system (HDFS).
- They provide a procedural data flow language with syntax and commands for implementing business logic.
- The scripts are converted into MapReduce jobs by the Pig Engine, abstracting the underlying MapReduce complexity from the user.

2. Structure:

- Pig Latin scripts consist of a series of statements that define data transformations.
- These statements include loading data, filtering, projecting, joining, grouping, and storing results.
- LOAD statements specify input data locations, format, and schema.
- STORE statements save the processed data, while DUMP statements display results on the command line.

3. Key Components:

- **Pig Latin Language**: The high-level language used for writing data analysis scripts.
- **Pig Engine**: The runtime engine that compiles Pig Latin scripts into MapReduce code.
- **HDFS**: The Hadoop Distributed File System where data is stored and retrieved.

4. Execution Flow:

- Pig Latin scripts are submitted to the Apache Pig server.
- The Pig Latin compiler parses, validates, and optimizes the script.
- The optimized script is then converted into a sequence of MapReduce jobs.
- These jobs are executed by the Pig Engine, leveraging the Hadoop cluster.

```
--load data
students = LOAD '/user/root/pigdata/students.txt'   USING
PigStorage(',')  AS (id:int, name:chararray, dept:chararray,
marks:int);

-- Filter students with marks > 80
high_scorers = FILTER students BY marks > 80;

-- Sort all students by marks descending
sorted_students = ORDER students BY marks DESC;

--Project only name and marks
projected = FOREACH students GENERATE name, marks;

-- Group by department and find average marks
grouped = GROUP students BY dept;
average_marks = FOREACH grouped GENERATE group AS department,
AVG(students.marks) AS avg_marks;

STORE sorted_students INTO '/user/root/pigoutput/sorted_students'
USING PigStorage(',');
STORE high_scorers INTO '/user/root/pigoutput/high_scorers' USING
PigStorage(',');
STORE projected INTO '/user/root/pigoutput/projected' USING
PigStorage(',');
STORE average_marks INTO '/user/root/pigoutput/average_marks' USING
PigStorage(',');
```

OUTPUT:

[cloudera@quickstart ~]$ hdfs dfs -mkdir /user/root/pigdata

[cloudera@quickstart ~]$ hdfs dfs -copyFromLocal students.txt /user/root/pigdata

[cloudera@quickstart ~]$ hdfs dfs -cat /user/root/pigdata/students.txt

INPUT DATA : students.txt

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/root/pigdata/students.txt
101,John,CS,85
102,Alice,IT,92
103,Bob,CS,76
104,David,EC,89
105,Eve,IT,67
106,john,AIML,88
107,George,EC,100
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ pig -x mapreduce PigExample.pig /user/root/pigdata/studets.txt

[cloudera@quickstart ~]$ hdfs dfs -ls /user/root/pigoutput

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$
[cloudera@quickstart Desktop]$ hdfs dfs -ls /user/root/pigoutput
Found 4 items
drwxr-xr-x   - cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/average_marks
drwxr-xr-x   - cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/high_scorers
drwxr-xr-x   - cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/projected
drwxr-xr-x   - cloudera supergroup          0 2025-08-06 00:16 /user/root/pigoutput/sorted_students
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -ls /user/root/pigoutput/projected

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -ls /user/root/pigoutput/projected
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/projected/_SUCCESS
-rw-r--r--   1 cloudera supergroup         59 2025-08-06 00:14 /user/root/pigoutput/projected/part-m-00000
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -ls /user/root/pigoutput/average_marks

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -ls /user/root/pigoutput/average_marks
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/average_marks/_SUCCESS
-rw-r--r--   1 cloudera supergroup         34 2025-08-06 00:14 /user/root/pigoutput/average_marks/part-r-00000
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -ls /user/root/pigoutput/high_scorers

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -ls /user/root/pigoutput/high_scorers
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2025-08-06 00:14 /user/root/pigoutput/high_scorers/_SUCCESS
-rw-r--r--   1 cloudera supergroup         82 2025-08-06 00:14 /user/root/pigoutput/high_scorers/part-m-00000
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -ls /user/root/pigoutput/sorted_students

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -ls /user/root/pigoutput/sorted_students
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2025-08-06 00:16 /user/root/pigoutput/sorted_students/_SUCCESS
-rw-r--r--   1 cloudera supergroup        110 2025-08-06 00:16 /user/root/pigoutput/sorted_students/part-r-00000
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -cat /user/root/pigoutput/average_marks/part-r-00000

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/root/pigoutput/average_marks/part-r-00000
CS,80.5
EC,94.5
IT,79.5
AIML,88.0
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -cat /user/root/pigoutput/high_scorers/part-m-00000

```
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/root/pigoutput/high_scorers/part-m-00000
101,John,CS,85
102,Alice,IT,92
104,David,EC,89
106,john,AIML,88
107,George,EC,100
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -cat /user/root/pigoutput/projected/part-m-00000

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/root/pigoutput/projected/part-m-00000
John,85
Alice,92
Bob,76
David,89
Eve,67
john,88
George,100
[cloudera@quickstart Desktop]$
```

[cloudera@quickstart ~]$ hdfs dfs -cat /user/root/pigoutput/sorted_students/part-r-00000

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/root/pigoutput/sorted_students/part-r-00000
107,George,EC,100
102,Alice,IT,92
104,David,EC,89
106,john,AIML,88
101,John,CS,85
103,Bob,CS,76
105,Eve,IT,67
[cloudera@quickstart Desktop]$
```

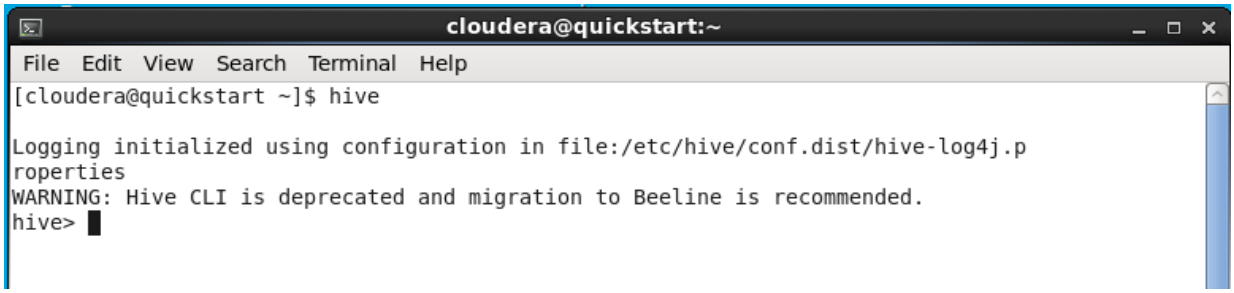
Program 7

Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

HiveQL or HQL is a Hive query language that we used to process or query structured data on Hive. HQL syntaxes are very much similar to MySQL but have some significant differences. We will use the hive command, which is a bash shell script to complete our hive demo using CLI(Command Line Interface). We can easily start hive shell by simply typing hive in the terminal.

Databases in Apache Hive

The Database is a storage schema that contains multiple tables. The Hive Databases refer to the namespace of tables. If you don't specify the database name by default Hive uses its default database for table creation and other purposes. Creating a Database allows multiple users to create tables with a similar name in different schemas so that their names don't match.



Create Database Syntax:

We can create a database with the help of the below command but if the database already exists then, in that case, Hive will throw an error.

```
CREATE DATABASE|SCHEMA <database name>    # we can use DATABASE or SCHEMA for
creation of DB
```

Example:

```
hive> CREATE DATABASE Test;
OK
Time taken: 0.15 seconds
hive> show databases;
OK
default
test
Time taken: 0.017 seconds, Fetched: 2 row(s)
hive>
```

If we again try to create a Test database hive will throw an error/warning that the database with the name Test already exists. In general, we don't want to get an error if the database exists. So we use the create database command with [IF NOT EXIST] clause. This will do not throw any error.

```
CREATE SCHEMA IF NOT EXISTS Test1;

SHOW DATABASES;
```

Example:

```
CREATE SCHEMA IF NOT EXISTS Test1;

SHOW DATABASES;
```

```
hive> CREATE SCHEMA IF NOT EXISTS Test1;
OK
Time taken: 0.024 seconds
hive> SHOW DATABASES;
OK
default
test
test1
Time taken: 0.013 seconds, Fetched: 3 row(s)
hive>
```

Syntax To Drop Existing Databases:

```
DROP DATABASE <db_name>;  or  DROP DATABASE IF EXIST <db_name>  # The IF EXIST
clause again is used to suppress error
```

Example:

```
DROP DATABASE IF EXISTS Test;

DROP DATABASE Test1;
```

```
hive> DROP DATABASE IF EXISTS Test;
OK
Time taken: 0.055 seconds
hive> DROP DATABASE Test1;
OK
Time taken: 0.033 seconds
hive> show databases;
OK
default
Time taken: 0.012 seconds, Fetched: 1 row(s)
hive>
```

Now quit hive shell with quit command.

```
quit;
```

```
hive> quit;
dikshant@dikshant:~$
```

Hive DDL Operations on Databases, Tables, Views, Functions, and Indexes

1. Databases

- **• Create a Database:**

CREATE DATABASE college_db;

- **• Use the Database:**

USE college_db;

- **• Alter the Database:**

ALTER DATABASE college_db SET DBPROPERTIES ('owner'='vijay');

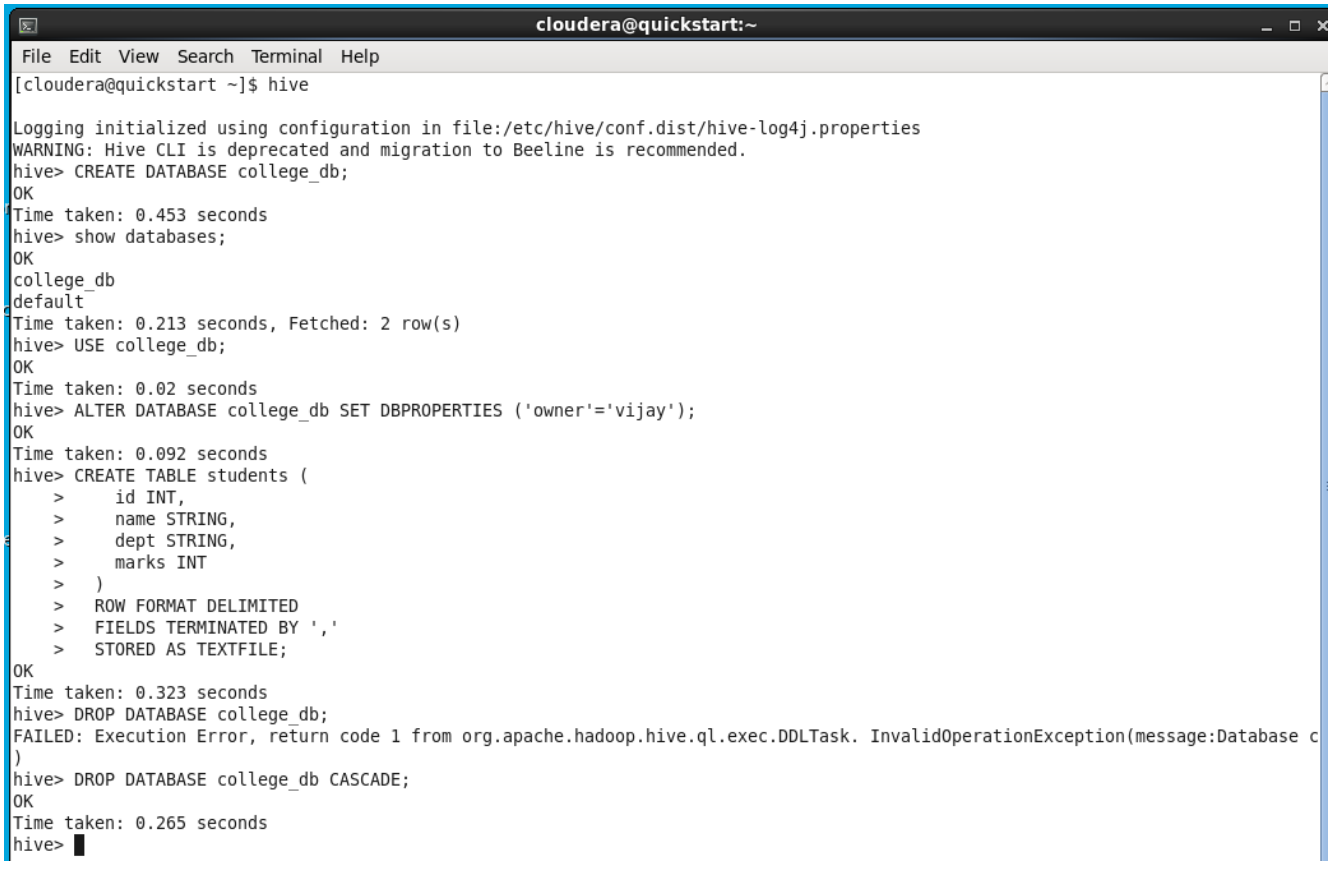
- **• Drop the Database:**

DROP DATABASE college_db;
DROP DATABASE college_db CASCADE; -- If it contains tables

2. Tables

- **• Create Table:**

```
CREATE TABLE students (
  id INT,
  name STRING,
  dept STRING,
  marks INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```



- **• Load Data into Table:**

LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/students.txt' INTO TABLE students;

- **• Alter Table (Add column):**

ALTER TABLE students ADD COLUMNS (email STRING);

- **• Drop Table:**

DROP TABLE students;

3. Views

- **• Create View:**

CREATE VIEW high_scorers AS SELECT name, marks FROM students WHERE marks > 80;

- **• Use/View it:**

SELECT * FROM high_scorers;

- **• Drop View:**

DROP VIEW high_scorers;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/students.txt' INTO TABLE students;
Loading data to table college_db.students
Table college_db.students stats: [numFiles=1, totalSize=156]
OK
Time taken: 0.433 seconds
hive> select * from Students;
OK
101     John    IS      85
102     Alice   IT      92
103     Bob     IS      76
104     David   EC      89
105     Eve     IT      67
106     john    AIML    88
107     George  EC      100
108     David   IS      99
109     Tim     IT      88
110     Clara   EC      97
Time taken: 0.431 seconds, Fetched: 10 row(s)
hive> ALTER TABLE students ADD COLUMNS (email STRING);
OK
Time taken: 0.124 seconds
hive> desc Students;
OK
id                      int
name                    string
dept                    string
marks                   int
email                   string
Time taken: 0.098 seconds, Fetched: 5 row(s)
hive>
```

```
hive> CREATE VIEW high_scorers AS SELECT name, marks FROM students WHERE marks > 80;
OK
Time taken: 0.134 seconds
hive> SELECT * FROM high_scorers;
OK
John    85
Alice   92
David   89
john    88
George  100
David   99
Tim     88
Clara   97
Time taken: 0.132 seconds, Fetched: 8 row(s)
hive> DROP VIEW high_scorers;
OK
Time taken: 0.149 seconds
hive>
```

4. Functions

- **• Built-in Function Example:**

SELECT UPPER(name) FROM students;

5. Indexes

- **• Create Index (Hive ≤ 3.x):**

CREATE INDEX student_idx
ON TABLE students (dept)
AS 'COMPACT'
WITH DEFERRED REBUILD;

```
hive> SELECT UPPER(name) FROM students;
OK
JOHN
ALICE
BOB
DAVID
EVE
JOHN
GEORGE
DAVID
TIM
CLARA
Time taken: 0.063 seconds, Fetched: 10 row(s)
hive> CREATE INDEX student_idx
    >   ON TABLE students (dept)
    >   AS 'COMPACT'
    >   WITH DEFERRED REBUILD;
OK
Time taken: 0.145 seconds
hive>
```

- **• Build Index:**

ALTER INDEX student_idx ON students REBUILD;

- **• Drop Index:**

DROP INDEX student_idx ON students;

```
hive> ALTER INDEX student_idx ON students REBUILD;
Query ID = cloudera_20250815224545_80cf5f12-6793-4eee-a56c-6d84874181d3
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_1755318511754_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application
_1755318511754_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1755318511754_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2025-08-15 22:46:04,467 Stage-1 map = 0%,  reduce = 0%
2025-08-15 22:46:13,228 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.2 sec
2025-08-15 22:46:24,139 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.85 sec
MapReduce Total cumulative CPU time: 2 seconds 850 msec
Ended Job = job_1755318511754_0003
Loading data to table college_db.college_db__students_student_idx__
Table college_db.college_db__students_student_idx__ stats: [numFiles=1, numRows=4, totalSize=398, rawDa
taSize=394]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.85 sec   HDFS Read: 9087 HDFS Write: 500 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 850 msec
OK
Time taken: 33.681 seconds
hive> DROP INDEX student_idx ON students;
OK
Time taken: 0.115 seconds
hive>
```

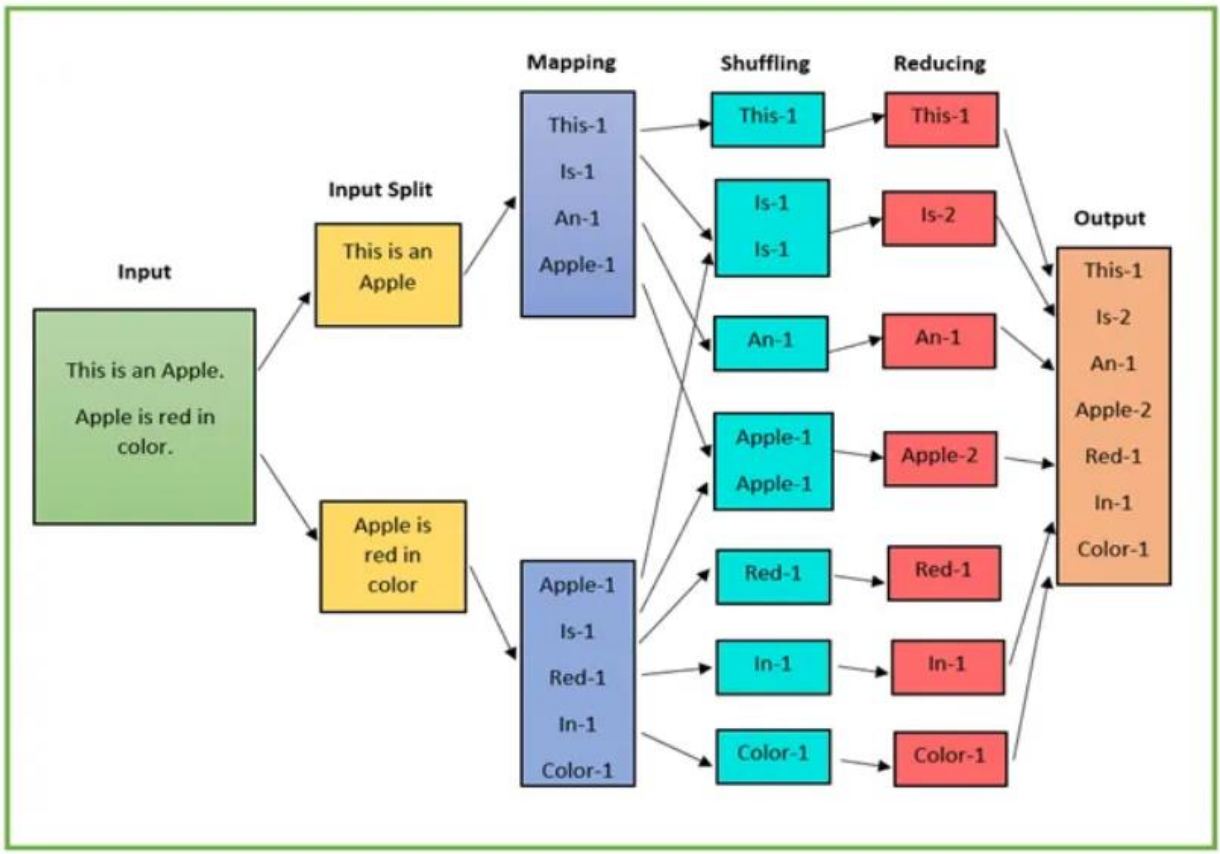
Program 8

Implement a word count program in Hadoop and Spark

MapReduce in Hadoop is a software framework for ease in writing applications of software, processing huge amounts of data. MapReduce provides the facility to distribute the workload (computations) among various nodes(analogous to commodity hardware). Hence, reducing the processing time as data on which the computation needs to be done is now divided into small chunks and individually processed. Through MapReduce you can achieve parallel processing resulting in faster execution of the job**.**

MapReduce Word Count is a framework which splits the chunk of data, sorts the map outputs and input to reduce tasks. A File-system stores the output and input of jobs. Re-execution of failed tasks, scheduling them and monitoring them is the task of the framework.

Figure below shows the architecture as well as working of MapReduce with an example:



Splitting: The parameter of splitter can be anything. By comma, space, by a new line or a semicolon.

Mapping: This is done as explained below.

Shuffle/Intermediate splitting: The process is usually parallel on cluster keys. The output of the map gets into the Reducer phase and all the similar keys of data are aligned in a cluster.
Reducing: This is done as explained below. Final result — All the data is clustered or combined to show the together form of a result.

Implementation a word count program in Hadoop

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
  public static void main(String [] args) throws Exception
  {
    Configuration c=new Configuration();
    String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
    Path input=new Path(files[0]);
    Path output=new Path(files[1]);
    Job j=new Job(c,"wordcount");
    j.setJarByClass(WordCount.class);
    j.setMapperClass(MapForWordCount.class);
    j.setReducerClass(ReduceForWordCount.class);
    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(j, input);
    FileOutputFormat.setOutputPath(j, output);
    System.exit(j.waitForCompletion(true)?0:1);
  }
public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
 IntWritable>{
  public void map(LongWritable key, Text value, Context con) throws
IOException, InterruptedException
  {
    String line = value.toString();
    String[] words=line.split(" ");
    for(String word: words )
    {
      Text outputKey = new Text(word.toUpperCase().trim());
      IntWritable outputValue = new IntWritable(1);
      con.write(outputKey, outputValue);
```

```
    }
  }
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable,
Text,
IntWritable>
{
  public void reduce(Text word, Iterable<IntWritable> values, Context con)
 throws IOException, InterruptedException
  {
    int sum = 0;
    for(IntWritable value : values)
    {
      sum += value.get();
    }
    con.write(word, new IntWritable(sum));
  }
 }
}
```

OUTPUT:

[cloudera@quickstart ~]$ hdfs dfs -mkdir /input_wordCount

[cloudera@quickstart ~]$ cd Desktop

[cloudera@quickstart ~]$ hdfs dfs -copyFromLocal word.txt /input_wordCount

[cloudera@quickstart ~]$ hdfs dfs -cat /input_wordCount/word.txt

[cloudera@quickstart ~]$ hdfs dfs -ls /input_wordCount

[cloudera@quickstart ~]$ hadoop jar WordCount.jar WordCount /input_wordCount /output_dir

[cloudera@quickstart ~]$ hdfs dfs -cat /output_dir/*

```
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart Desktop]$ hdfs dfs -cat /output_dir/part-r-00000
BIT     1
CODE    1
HE      1
IN      1
IS      2
LIKES   1
MY      1
NAME    1
TO      1
VIJAY   2
WORKING 1
[cloudera@quickstart Desktop]$
```

Program 9

Use CDH (Cloudera Distribution for Hadoop) and HUE (Hadoop User Interface) to analyze data and generate reports for sample datasets.

Cloudera Distribution Hadoop (CDH) was a popular open-source distribution of Apache Hadoop and related projects, designed for enterprise-level deployments. It offered a unified platform for storing and analyzing large datasets, integrating various components like HDFS, MapReduce, YARN, Spark, Hive, HBase, and more. Cloudera has since moved away from CDH and now offers the Cloudera Data Platform (CDP) which combines the strengths of CDH and Hortonworks Data Platform (HDP).

Key Features and Components of CDH:

- **Apache Hadoop Core:**
  CDH included core Hadoop components like HDFS (for distributed storage), MapReduce (for parallel processing), and YARN (for resource management).
- **Ecosystem Integration:**
  It integrated various other Apache projects to extend Hadoop's functionality, such as Spark (for fast data processing), Hive (for SQL-like querying), HBase (for NoSQL database), and more.
- **Cloudera Manager:**
  A management console for easy deployment, configuration, monitoring, and management of CDH clusters.
- **Enterprise-Ready:**
  CDH was designed for enterprise environments, providing features like security, high availability, and commercial support.
- **SQL-on-Hadoop:**
  Cloudera was a pioneer in SQL-on-Hadoop with its Impala query engine.
- **Node Templates:**
  CDH allowed for the creation of node templates with varying configurations within a Hadoop cluster, eliminating the need for uniform configurations.

Hue is a web-based interactive query editor that enables you to interact with databases and data warehouses. Data architects, SQL developers, and data engineers use Hue to create data models, clean data to prepare it for analysis, and to build and test SQL scripts for applications.

Hue offers powerful execution, debugging, and self-service capabilities to the following key Big Data personas:

Business Analysts
Data Engineers
Data Scientists
Power SQL users
Database Administrators
SQL Developers

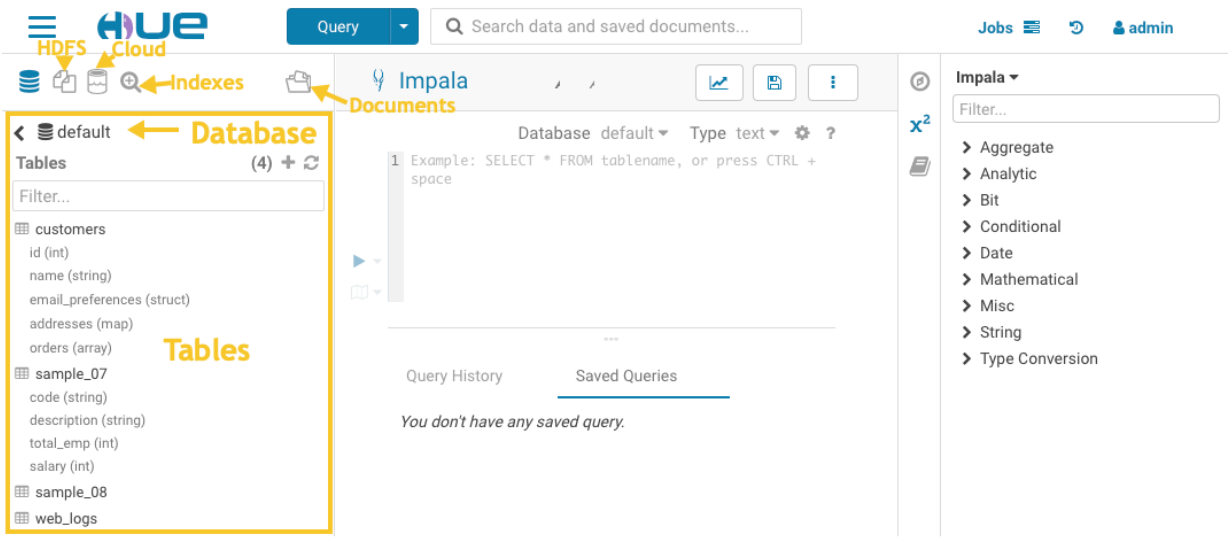
Steps to execute the program:

1. Create a table sales_data having following fields(id, date, region, product, qty, price, sales).
2. Query the data and visualize.

1. Create the table.

Open **Hue (**Hue is a web-based interactive query editor that enables you to interact with databases by running query).

You can use Hue to: **Explore, browse, and import your data** through guided navigation in the left panel of the page:



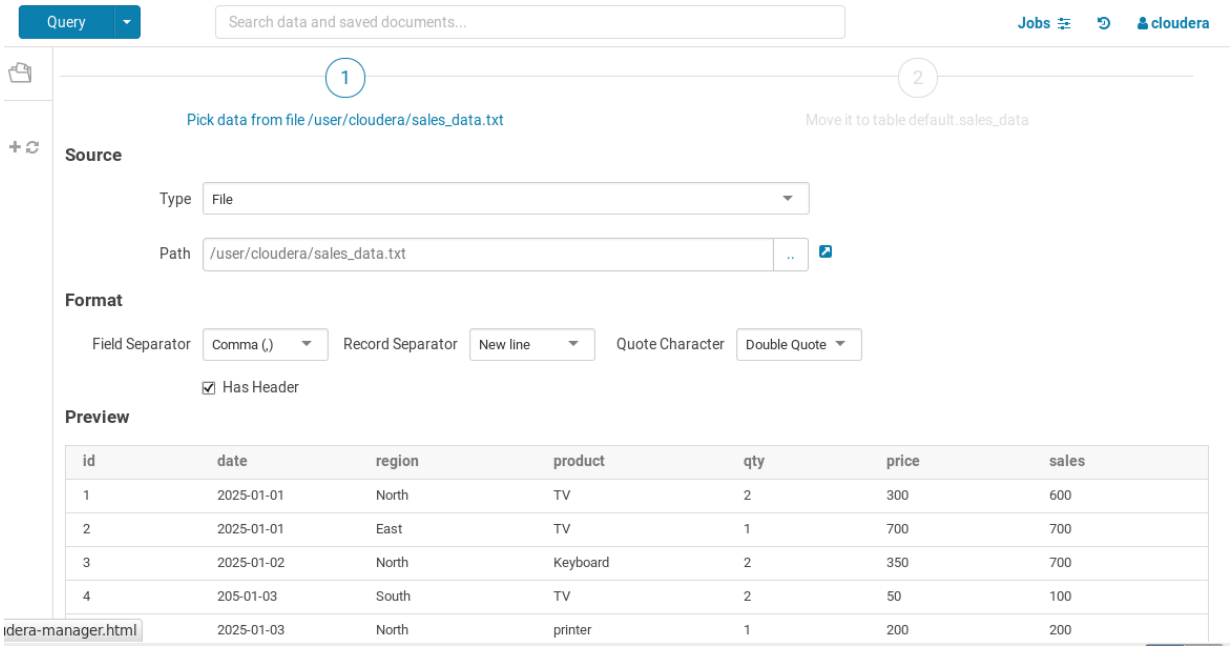
Initially panel displays 2 options

1. Impala

2. Hive as shown

- Select Hive
- Click on '+' to create tables under the Databases > Tables
- Under SOURCE, select Remote File from the Type drop-down menu.
- Click .. at the end of the Path field.
- The Choose a file modal is displayed.
- Browse and select the file you want to use to create a table. Hue displays the preview of the table along with the format.

  The following screenshot is the results of the above steps.



- Click **Next**.
- The table destination and properties are displayed.
- **Optional:** Set the table destination, partitions, and change the column data types.
- Verify the settings and click **Submit** to create the table.

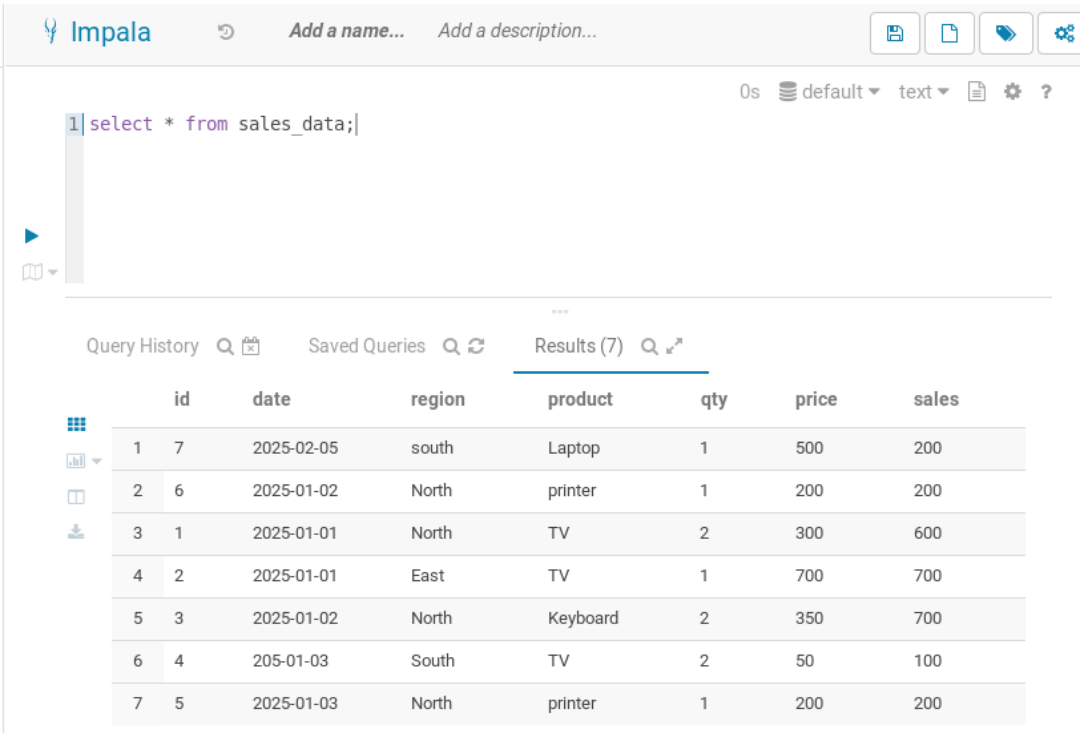
  The CREATE TABLE query is triggered.

Hue displays the logs and opens the Table Browser from which you can view the newly created table when the operation completes successfully.

2. Query and Visualize the data

To run a query:

- Click a database to view the tables it contains.
- When you click a database, it sets it as the target of your query in the main query editor panel.
- Type a query in the editor panel and click play button (to run the query).
- You can also run multiple queries by selecting them and clicking.

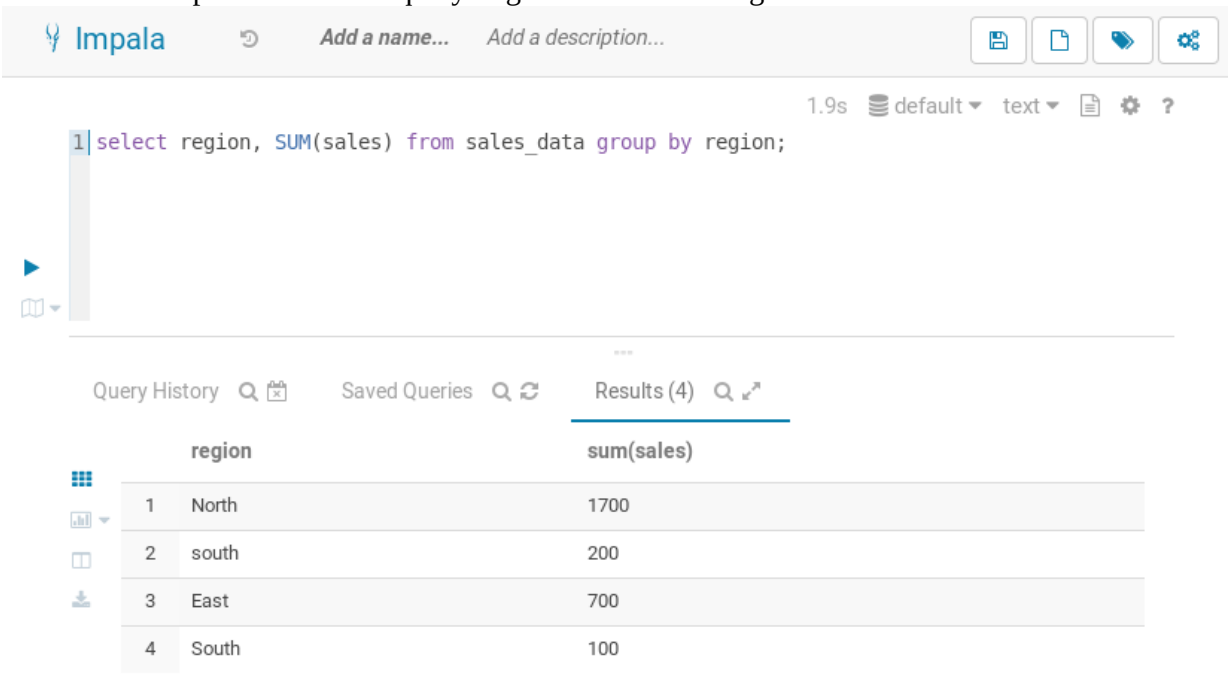


	id	date	region	product	qty	price	sales
1	7	2025-02-05	south	Laptop	1	500	200
2	6	2025-01-02	North	printer	1	200	200
3	1	2025-01-01	North	TV	2	300	600
4	2	2025-01-01	East	TV	1	700	700
5	3	2025-01-02	North	Keyboard	2	350	700
6	4	205-01-03	South	TV	2	50	100
7	5	2025-01-03	North	printer	1	200	200

3. Reports and Charts in HUE

SQL Developers can use Hue to create data sets to generate reports and dashboards that are often consumed by other Business Intelligence (BI) tools, such as Cloudera Data Visualization.

- Edit the query in the Query Editor.
- Run the query.
- Click the button below list (icon) button as shown in above screenshot.
- The output chart for the query to get the total sales region wise is shown below.



	region	sum(sales)
1	North	1700
2	south	200
3	East	700
4	South	100

- The chart got on clicking chart icon just below the list if icons in the list of icons. Set the x-axis and y-axis fields.