



LAB REPORT

**Submitted By:**

**Name: Abhishek Kumar**

**Reg no: 24MSD7017**

**Submitted To : Prof Avadesh Kumar**

**● INTRODUCTION TO NEO4J**

**Neo4j**  
Neo4j is a highly popular **graph database management system (GDBMS)** designed to efficiently store, manage, and query complex, highly connected data. Unlike traditional relational databases that use tables, Neo4j represents data as **nodes, relationships, and properties**, making it ideal for understanding and analyzing the relationships between entities. It is widely used in domains such as social network analysis, fraud detection, recommendation systems, and knowledge graphs.

Here are some key points about Neo4j:

1. **Graph-Based Data Model:**  
   Neo4j uses a property graph model, where **nodes** represent entities (e.g., people, places, products), **relationships** represent connections between nodes (e.g., “FRIENDS\_WITH”, “PURCHASED”), and both can have **properties** (key-value pairs).
2. **Cypher Query Language:**  
   Neo4j uses **Cypher**, a declarative query language similar to SQL but designed specifically for graph data. Cypher allows users to easily express complex graph patterns and relationships.
3. **High Performance on Connected Data:**  
   Neo4j is optimized for traversing relationships, making queries involving multiple connections extremely fast compared to relational databases.
4. **Scalability and Flexibility:**  
   It can handle datasets ranging from small networks to massive, enterprise-scale graphs, supporting horizontal scaling through Neo4j AuraDB (cloud version).
5. **Real-World Applications:**  
   Neo4j is used in diverse fields such as **recommendation engines, fraud detection, supply chain optimization, knowledge graphs, network management**, and **semantic search**.
6. **ACID Compliance:**  
   Neo4j ensures **data integrity and reliability** with full ACID (Atomicity, Consistency, Isolation, Durability) compliance.
7. **Integration and Connectivity:**  
   It integrates with popular programming languages (Python, Java, JavaScript, etc.) and data platforms. Neo4j also supports integration with big data ecosystems, visualization tools (like Tableau), and machine learning pipelines.

**Neo4j Desktop**  
**Neo4j Desktop** is the primary development interface for working with local and remote Neo4j databases. It provides a user-friendly environment for managing databases, running queries, and visualizing graph data interactively.

Here are some key features of Neo4j Desktop:

1. **Intuitive Interface:**  
   Neo4j Desktop offers a simple and interactive interface to create, connect, and manage multiple graph projects easily.
2. **Built-in Cypher Editor:**  
   Users can write, execute, and test Cypher queries directly in the built-in editor with syntax highlighting and query history.
3. **Graph Visualization:**  
   The tool provides visual graph exploration, enabling users to view relationships, filter nodes, and interact with the data dynamically.
4. **Plugin Support:**  
   Neo4j Desktop supports plugins and extensions for data science, graph algorithms, and integration with external tools.
5. **Local and Remote Connectivity:**  
   It allows users to connect to both local and remote Neo4j instances, including cloud databases like **Neo4j AuraDB**.
6. **Data Import and Export:**  
   Users can import CSV files or connect external data sources for building and updating graph databases.
7. **Collaboration and Sharing:**  
   Graph models and queries created in Neo4j Desktop can be shared with teams or deployed to production environments through Neo4j Server or AuraDB.

A black background with a black square

AI-generated content may be incorrect.

**REFERENCES**

* <https://neo4j.com/>
* <https://neo4j.com/product/neo4j-desktop/>

● **INTRODUCTION TO MONGODB**

**MongoDB**  
**MongoDB** is a leading **NoSQL, document-oriented database** designed to store, manage, and retrieve large volumes of unstructured or semi-structured data efficiently. Unlike traditional relational databases that store data in tables and rows, MongoDB stores data in **flexible JSON-like documents** (known as BSON – Binary JSON). This approach provides developers with high flexibility, scalability, and speed, making MongoDB a popular choice for modern web applications, analytics, and big data solutions.

Here are some key points about MongoDB:

1. **Document-Oriented Data Model:**  
   MongoDB stores data as **documents** (collections of key-value pairs), allowing each record to have a different structure. This makes it easy to handle evolving data models without requiring schema changes.
2. **NoSQL Database:**  
   As a NoSQL database, MongoDB does not rely on rigid table-based schemas. It supports dynamic and hierarchical data storage, which is ideal for complex and rapidly changing datasets.
3. **Scalability and Performance:**  
   MongoDB supports **horizontal scaling** through a technique called **sharding**, enabling it to handle very large datasets and high-throughput applications efficiently.
4. **High Availability:**  
   MongoDB provides **replica sets** that ensure automatic failover and data redundancy, maintaining high availability even during hardware failures.
5. **Rich Query Language:**  
   MongoDB offers a powerful query language that supports CRUD operations (Create, Read, Update, Delete), aggregation, indexing, and filtering—similar to SQL but more flexible for document-based data.
6. **Integration and Compatibility:**  
   MongoDB integrates easily with popular programming languages like Python, Java, C++, Node.js, and frameworks such as Django, Flask, and Express.js. It also works well with cloud platforms and analytics tools.
7. **Use Cases:**  
   MongoDB is widely used for **content management systems, IoT applications, real-time analytics, e-commerce platforms, mobile applications**, and **big data processing**.

**MongoDB Compass**  
**MongoDB Compass** is the official **graphical user interface (GUI)** for MongoDB, designed to help users visualize, explore, and analyze data stored in MongoDB databases. It allows developers and analysts to interact with data intuitively without writing complex queries.

Here are some key features of MongoDB Compass:

1. **User-Friendly Interface:**  
   MongoDB Compass provides an easy-to-navigate interface for browsing databases, collections, and documents.
2. **Schema Visualization:**  
   It automatically analyzes collections and provides a graphical view of the data schema, including field types, data distribution, and structure.
3. **Query Builder:**  
   Users can build, test, and run queries using a visual query builder without needing to write code, making it ideal for beginners.
4. **Performance Analysis:**  
   Compass allows users to analyze query performance and indexing, helping optimize database operations for speed and efficiency.
5. **CRUD Operations:**  
   Users can easily create, edit, and delete documents directly through the interface, simplifying database management tasks.
6. **Aggregation Pipeline Builder:**  
   It includes tools to design and execute **aggregation pipelines** for advanced data analysis and transformation.
7. **Security and Connection Management:**  
   MongoDB Compass supports secure connections and authentication mechanisms to safely connect to local and remote MongoDB servers.

A black background with blue letters

AI-generated content may be incorrect.

**REFERENCES**

* <https://www.mongodb.com/>
* <https://www.mongodb.com/products/compass>

● **INTRODUCTION TO HADOOP**

**Hadoop**  
**Apache Hadoop** is an **open-source framework** that enables the distributed storage and processing of large datasets across clusters of computers using simple programming models. It is designed to handle **big data**—data that is too large, complex, or fast-moving for traditional data processing systems. Hadoop provides a reliable, scalable, and cost-effective solution for managing and analyzing vast amounts of data, making it a cornerstone technology in the field of **Big Data Analytics**.

Here are some key points about Hadoop:

1. **Distributed Computing Framework:**  
   Hadoop divides large datasets into smaller chunks and distributes them across multiple nodes in a cluster. Each node processes its portion of data in parallel, resulting in high efficiency and fault tolerance.
2. **Open-Source and Scalable:**  
   Developed by the Apache Software Foundation, Hadoop is open-source and highly scalable—capable of running on a few machines or thousands of nodes.
3. **Fault Tolerance:**  
   Hadoop automatically replicates data across multiple nodes to ensure reliability. If a node fails, the system continues processing using replicated data without data loss.
4. **Cost-Effective Storage:**  
   Hadoop uses **commodity hardware** (low-cost machines) to store and process big data, making it an affordable solution for enterprises managing large-scale data.
5. **Batch Processing:**  
   Hadoop is designed primarily for **batch processing** of large datasets rather than real-time analytics. It excels in processing massive data sets efficiently.
6. **Ecosystem Integration:**  
   Hadoop integrates with a rich ecosystem of tools like **Hive, Pig, HBase, Spark, Flume, Sqoop, and Oozie**, which extend its functionality for data warehousing, real-time processing, and data ingestion.
7. **Use Cases:**  
   Hadoop is widely used in **data warehousing, log analysis, recommendation systems, fraud detection, market trend analysis**, and **scientific data processing**.

**Hadoop Ecosystem Components**  
The Hadoop ecosystem consists of several core and supporting components that work together to store, process, and manage data efficiently.

Here are the key components of the Hadoop ecosystem:

1. **Hadoop Distributed File System (HDFS):**  
   HDFS is the **storage layer** of Hadoop that stores large files across multiple machines. It divides data into blocks and distributes them across the cluster for parallel processing.
2. **Yet Another Resource Negotiator (YARN):**  
   YARN is the **resource management layer** of Hadoop that manages and schedules computing resources across the cluster for various applications.
3. **MapReduce:**  
   MapReduce is the **data processing framework** of Hadoop. It divides a job into two main tasks—**Map** (processing and filtering data) and **Reduce** (aggregating the results)—to process data in parallel.
4. **Apache Hive:**  
   Hive is a data warehouse tool built on Hadoop that allows users to perform **SQL-like queries (HiveQL)** on large datasets stored in HDFS.
5. **Apache Pig:**  
   Pig provides a high-level scripting language called **Pig Latin** for analyzing large data sets, simplifying the development of MapReduce programs.
6. **HBase:**  
   HBase is a **NoSQL database** that runs on top of HDFS, allowing real-time read and write access to large datasets.
7. **Apache Spark:**  
   Spark is a **fast, in-memory data processing engine** that extends Hadoop’s capabilities for real-time and iterative computations.

A blue and black logo

AI-generated content may be incorrect.

**REFERENCES**

* <https://hadoop.apache.org/>
* <https://hadoop.apache.org/docs/>

Contents

[Introduction 3](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.y52gz04lawcl)

[Experiments 4](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.61kccl6ywwr)

[Neo4j 4](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.40exqovddvqb)

[Introduction to Neo4j 4](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.ie9149otoorx)

HR Survey of Employees12

[Understanding Airlines Data Using Neo4j](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.if072bfqoy6s) 22

[MongoDB](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.gz1ym75pt7yi)

Analysis of Books in Library30

Find the No. of persons………………………………………………..39

[Hadoop (Cloudera)](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.mnj73hr8vi6)

[Introduction To Cloudera 41](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.yco6ggxn0x1j)

[Introduction To MapReduce](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.dca1g0iy28jg) 42

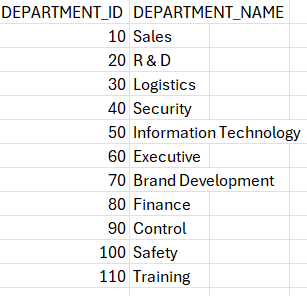
[Introduction To Hbase 4](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.imkm09bjcny8)4

[Conclusion 69](https://docs.google.com/document/d/1Z_DpREQ0S35jJ4cOMWmZ1RZrNxX1sBj1/edit#heading=h.kdas5hrj1vrh)

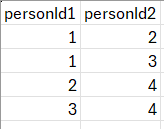
**LAB -1 HR Survey of Employees**

We have the following csv files

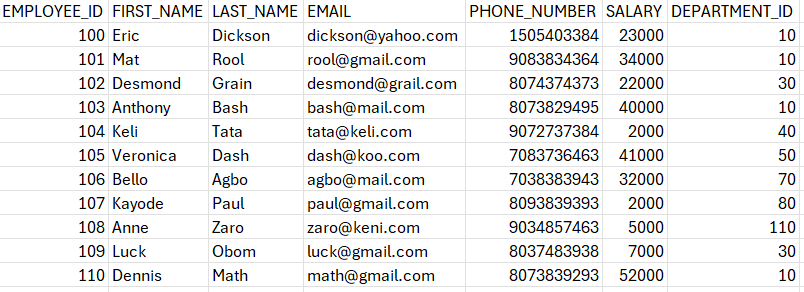
**#** Department.csv



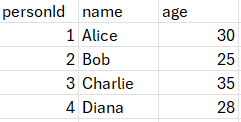
# Friendship.csv



# Employees.csv



# Person.csv



We have given the dataset (link) :

* 1. Department.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/1/files/departments.csv>
  2. Employees.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/1/files/employees.csv>
  3. Friendship.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/1/files/friendships.csv>
  4. Persons.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/1/files/persons.csv>

**# import persons.csv and friendship.csv**

1. Find mutual friends of Alice and Bob

2. Count number of friends each person has

3. Find the person with the most mutual friends

4. List of people older than average age

5. Find the shortest friendship path between Alice and Diana

**# import employees.csv and departments.csv**

1. List of all employees with their department

2. Show employee hierarchy (who manages whom)

3. Find departments with average salary

4. Find employees earning above the average salary

5. Find the top-level managers (who are not managed by anyone)

**# Importing the persons.csv and friendship.csv data using csv**A computer screen shot of a program code

AI-generated content may be incorrect.

**# Queries**

**1. Find mutual friends of Alice and Bob.**A screen shot of a computer

AI-generated content may be incorrect.

**Interpretation**: we can see Alice and Bob do not have mutual friends as per data.

**2. Count number of friends each person has.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Interpretation**: The table is sorted to show the person with the **most friends as the top.**

**3. Find the person with the most mutual friends**A screen shot of a computer

AI-generated content may be incorrect.

**Interpretation**: The query finds the pair of people with the most mutual friends, showing that **Bob** and **Charlie** share **2 mutual friends**.

**4. List of people older than average age**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Interpretation:** Charlie and Alice are older than the average age of all people in the database**.**

**5. Find the shortest friendship path between Alice and Diana**

A screen shot of a computer

AI-generated content may be incorrect.

**Interpretation:** The shortest friendship path between Alice and Diana is obtained as

2.

# importing **employees.csv** and **departments.csv**

A computer screen shot of a program

AI-generated content may be incorrect.

1. List of all employees with their departmentA screenshot of a computer program

AI-generated content may be incorrect.

**Interpretation:** The query lists all employees with their department and salary, ordered by department name and then employee’s last name.

**2. Show employee hierarchy (who manages whom)**

# Note: The provided data doesn't include manager relationships

# This query assumes we need to create a MANAGES relationship first

#Based on typical organizational structure, we can infer some relationships

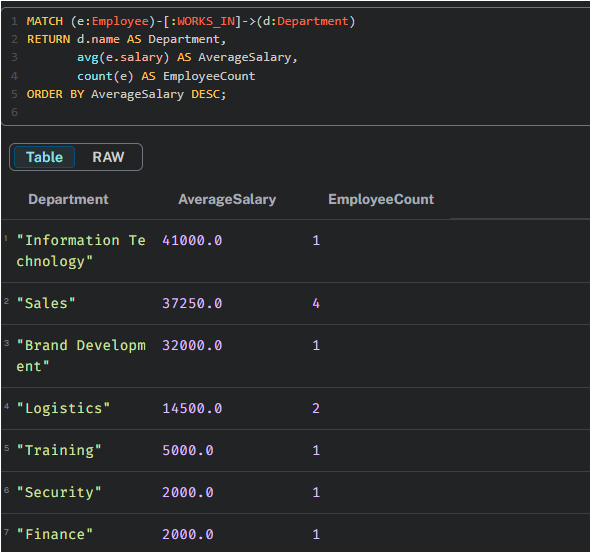
A screenshot of a computer program

AI-generated content may be incorrect.

**# Showing the Hierarchy**A screenshot of a computer

AI-generated content may be incorrect.

**Interpretation:** The output shows manager–subordinate relationships, where Dennis Math manages Eric Dickson, Mat Rool, and Anthony Bash, while Desmond Grain manages Luck Obom.

**3. Find departments with average salary**

**Interpretation:** The output lists each department with its average salary and employee count, sorted by highest average salary.

**4. Find employees earning above the average salary**

A screen shot of a computer

AI-generated content may be incorrect.

**Interpretation:** The output lists employees earning above the overall average salary (23,636.36), along with their department, ordered by salary in descending order.

**5. Find the top-level managers (who are not managed by anyone)**A screenshot of a computer screen

AI-generated content may be incorrect.

**// This query assumes MANAGES relationships have been created**A screenshot of a computer program

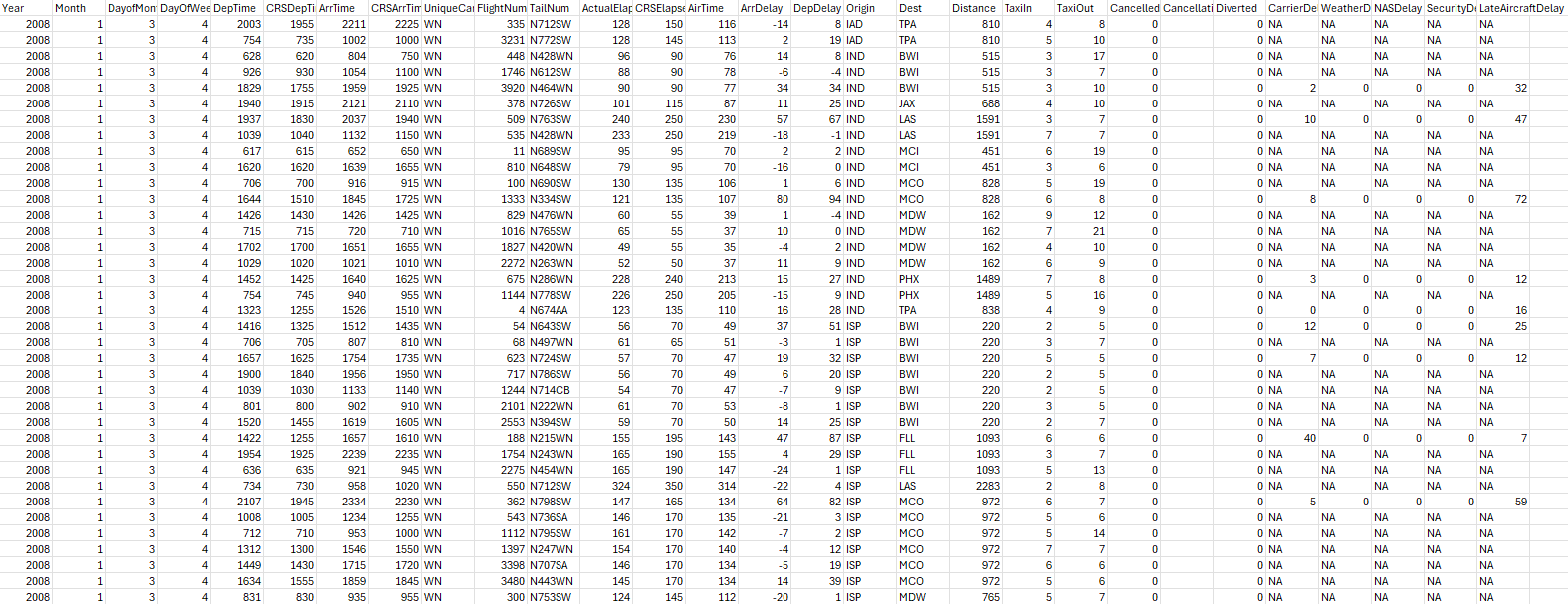
AI-generated content may be incorrect.

**Interpretation:** The output shows the highest-paid employee in each department, sorted by salary in descending order.

**LAB 2 Flight Survey Data**

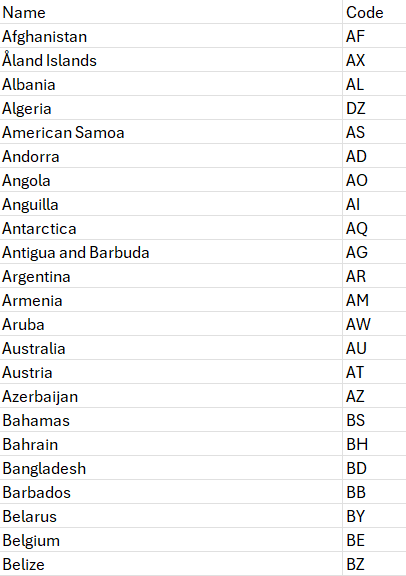
We have given the dataset of Flight Survey Data:

# flights.csv

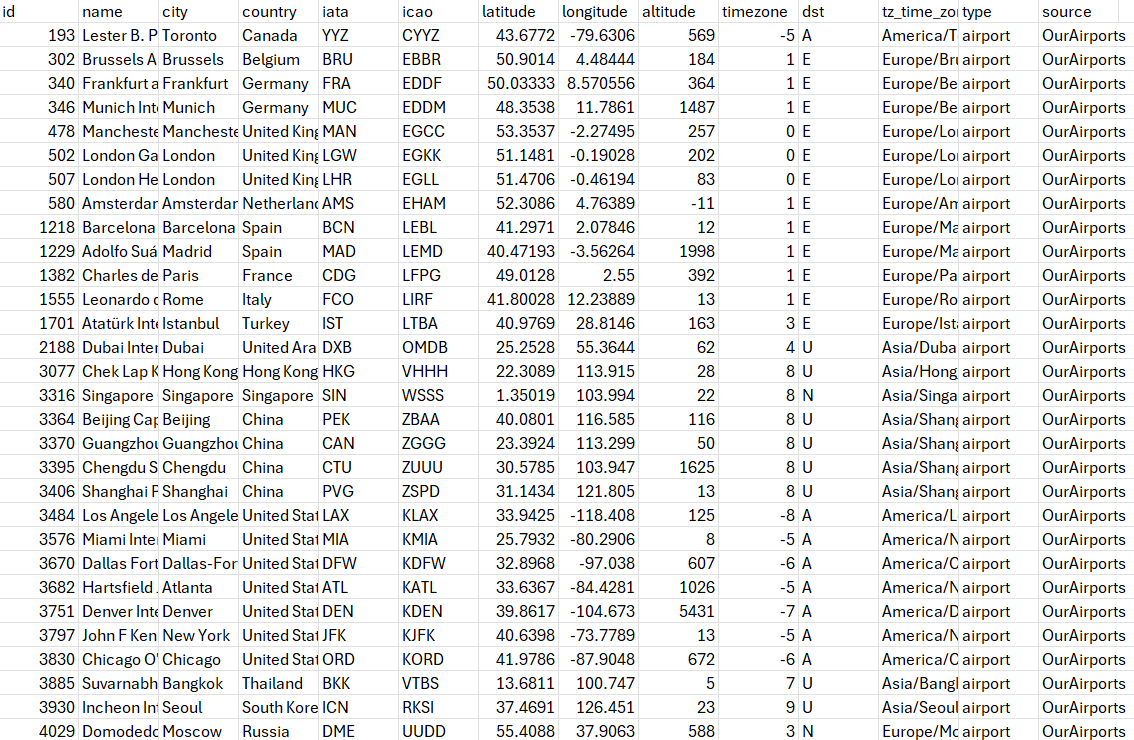


Which contains 10000 rows and 29 columns

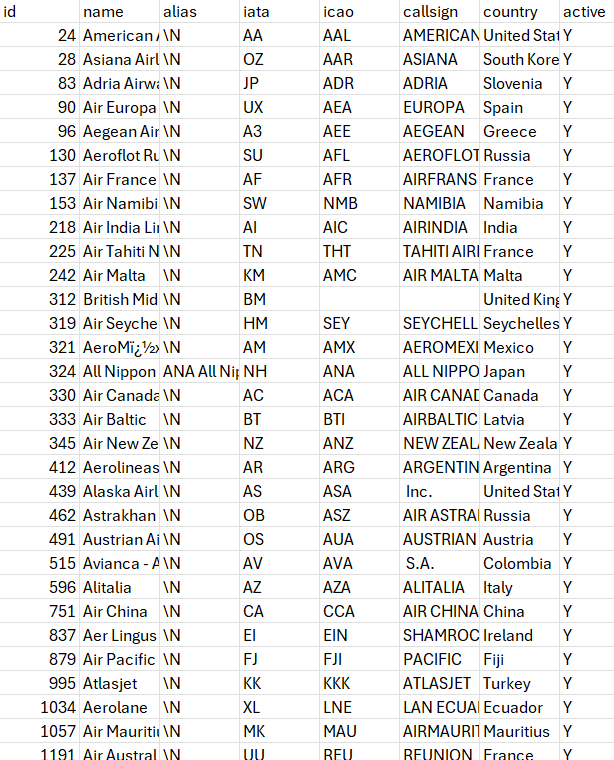
# Countries.csv



# airport.csv



# airlines.csv



Dataset link:-

* + 1. Flight.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/2/files/flights.csv>
    2. Country.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/2/files/countries.csv>
    3. Airport.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/2/files/airports.csv>
    4. Airlines.csv : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/2/files/airlines.csv>

// APOC-based import script for airlines, airports, countries, flights

// This version avoids USING PERIODIC COMMIT and works inside explicit transactions.

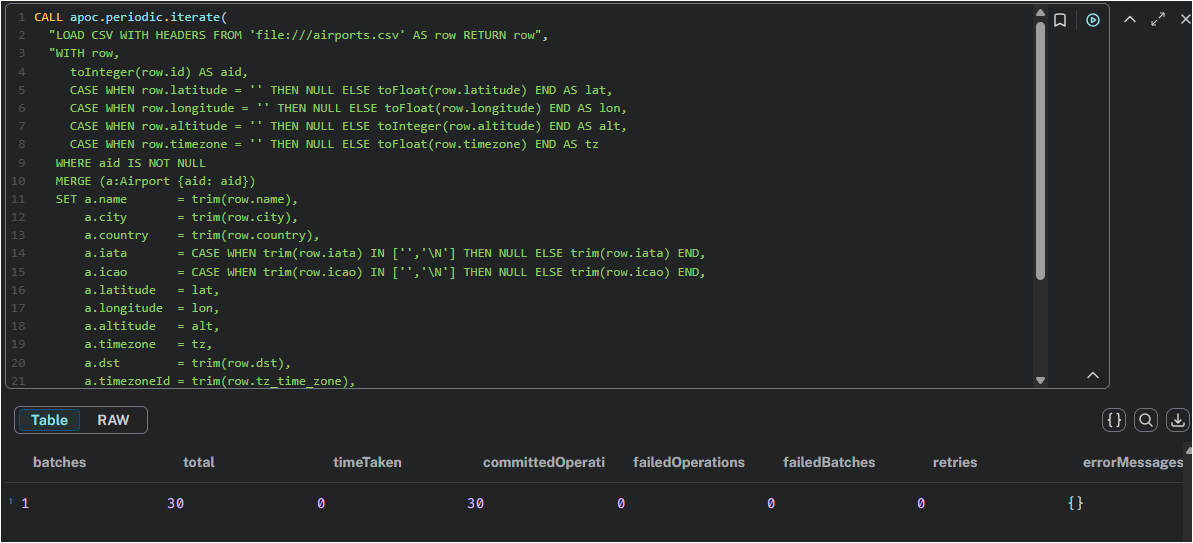
// Prereqs: APOC core installed & enabled(dbms.security.procedures.unrestricted=apoc.\*).

// Place CSVs in Neo4j's import directory and adjust filenames if needed.A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



A screen shot of a computer

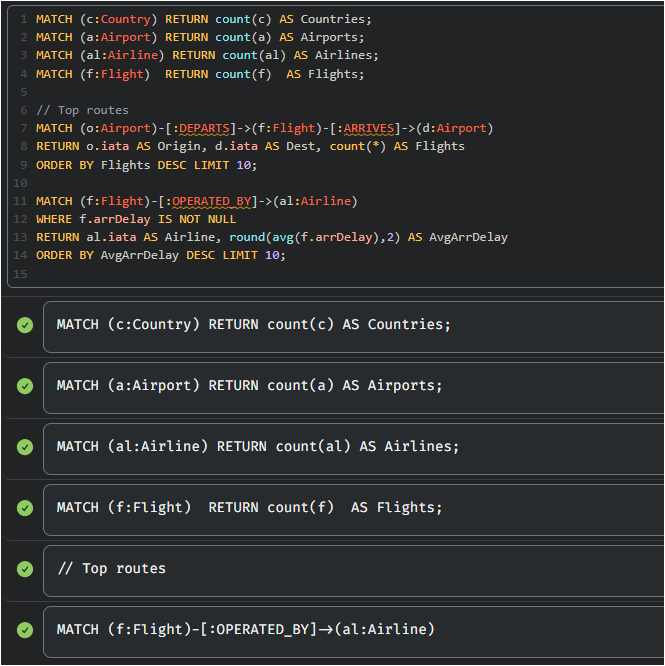
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



# **Queries with answers**

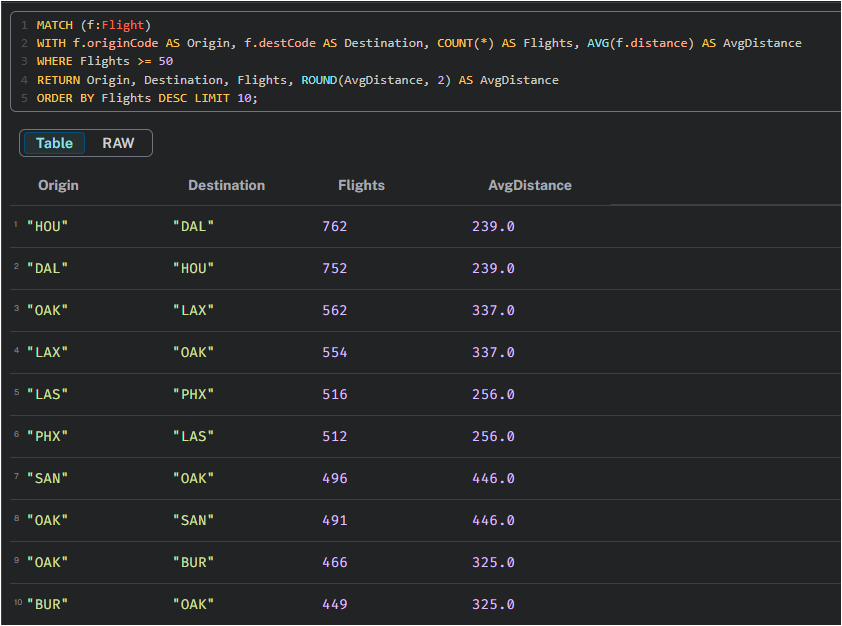
**Q1** **Count Countries, Airports, Airlines, Flights**

A screenshot of a computer program

AI-generated content may be incorrect.

**Interpretation:**

1. The database has global coverage (249 countries).
2. A relatively small number of airports (30), possibly filtered or limited
3. A large number of airlines (139), indicating diverse operators.
4. A very high number of flights (100,000), which could represent historical or scheduled data.

**Q2. Top 10 busiest routes by number of flights.**

**Interpretation:**

The query reveals the top 10 busiest U.S. flight routes—mostly short-haul connections—with Houston, Dallas, and Oakland emerging as key hubs.

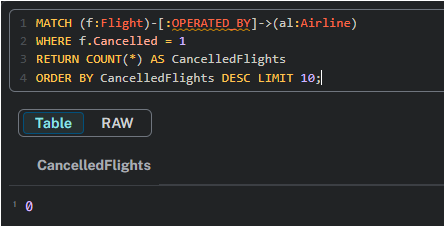
**Q3. Busiest days by total flights.**

A screenshot of a computer

AI-generated content may be incorrect.

**Interpretation:**

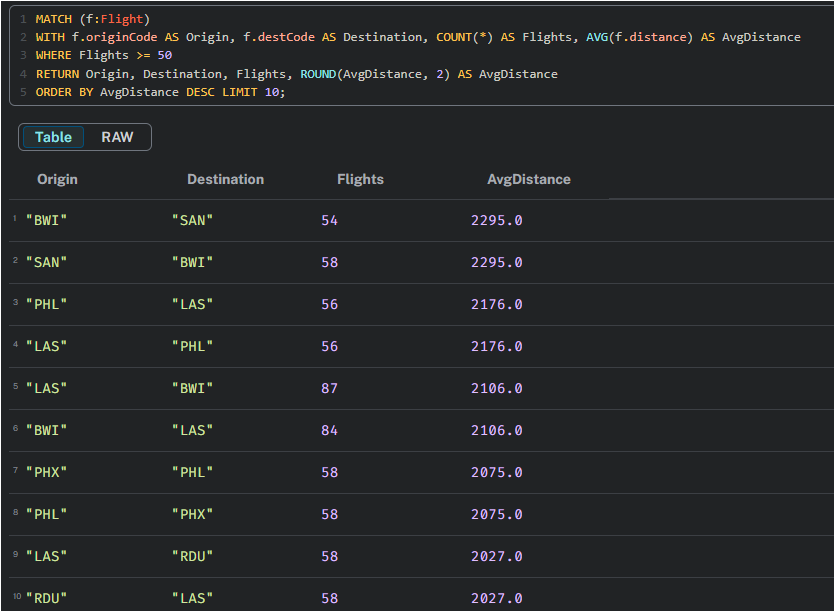
The query identifies the top 10 dates with the highest number of flights in January 2008, showing that flight traffic was consistently high peaking at 3,639 flights on January 31.

**Q4. Airlines with most cancellations**

**Interpretation:**

The query attempts to identify the top 10 airlines with the most canceled flights, but the result shows zero cancellations—suggesting either the dataset lacks cancellation data or all flights were completed without cancellations

**Q5. Longest average-distance routes (≥ 50 flights on the OD pair)**



**Interpretation:**

The query identifies the top 10 longest average-distance flight routes (with at least 50 flights), revealing that Baltimore (BWI), San Diego (SAN), Philadelphia (PHL), and Las Vegas (LAS) are key endpoints for long-haul domestic travel.

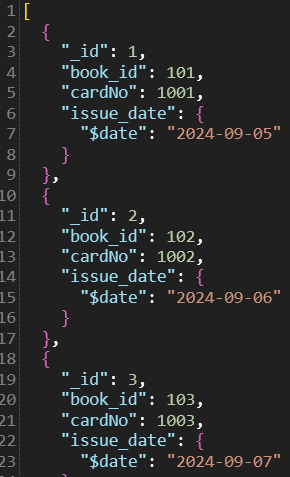
**Lab -3 Analysis of Books in library**

Given the dataset :

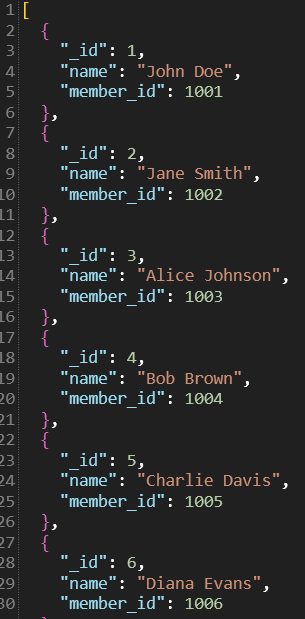
# Books.json



# issues.json



# members.json



Dataset link:

* + - 1. Books.json : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/3/files/books.json>
      2. Issues.json : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/3/files/issues_fixed.json>
      3. Members.json : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/3/files/members.json>

After importing the Json files in mongodb

Based on the following dataset and the question give below in using mongodb

**1. Find the Most Frequent Author in a Books Collection**

Given a collection of books, each book has multiple authors. Write a query to find the author who has written the most books.

**Collection Example (books):**

{

"\_id": 1,

"title": "Book One",

"authors": ["Author A", "Author B"]

},

#### Solution:

We can use the aggregation pipeline to unwind the authors, group them, and then sort to find the most frequent author.

db.books.aggregate([

{ $unwind: "$authors" },

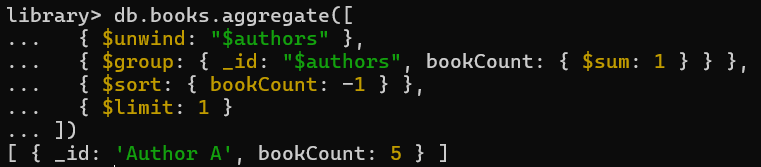
{ $group: { \_id: "$authors", count: { $sum: 1 } } },

{ $sort: { count: -1 } },

{ $limit: 1 }

]);

Answer is



**Explanation:**

* **$unwind** turns "authors": ["Alice", "Bob"] into two separate docs: one with "Alice", one with "Bob".
* **$group** counts how many times each author appears across all books.
* **$sort** arranges authors by their count (highest first).
* **$limit: 1** gives you just the **most frequent author**.

### 2. Find Books Issued in the Last Month

Given a collection of books and issues, find all books that were issued within the last 30 days.

#### Collection Example (issues):

{

"\_id": 1,

"book\_id": 101,

"issue\_date": ISODate("2024-09-05")

},

{

"\_id": 2,

"book\_id": 102,

"issue\_date": ISODate("2024-08-15")

}

#### Solution:

We can filter based on the issue\_date field using the $gte operator and new Date() to get the current date.

db.issues.find({

issue\_date: {

$gte: new Date(new Date().setDate(new Date().getDate() - 30))

}

});

**Explanation:**

* new Date(): Retrieves the current date.
* $gte: Ensures the issue\_date is greater than or equal to 30 days ago.

### 3. Top N Most Popular Books by Number of Issues

Find the top N most issued books from the issues collection.

### Answer

### 

Conclusion:

The query correctly grouped all these null-value documents, counted them (resulting in issueCount: 20), and returned that group as the top result. No other documents with actual book titles were found.

### 4. Find All Members Who Have Not Issued Any Books

Find all members from the members collection who have not issued any books from the issues collection.

#### Collection Example (members):

#### {

#### "\_id": 1,

#### "name": "John Doe",

#### "member\_id": 1001

#### },

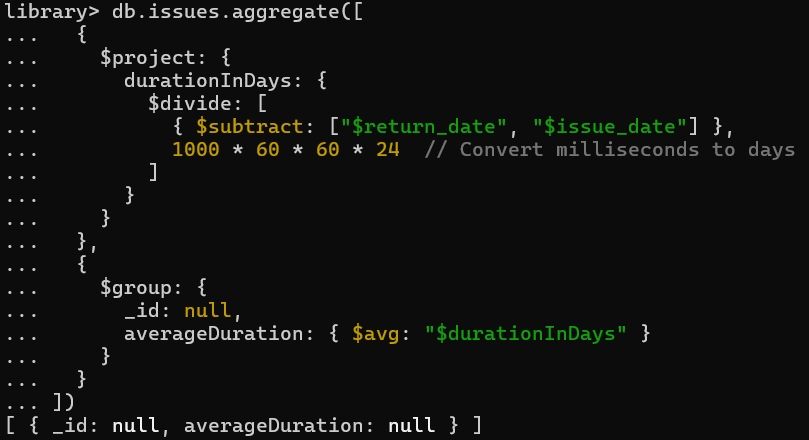
**Answer**

****

**5. Calculate Average Issue Duration for Each Book**

Given a collection of issues with issue\_date and return\_date, calculate the average duration a book was issued for.

Answer



Conclusion:

Because Data Don’t have the issue time so here, we are getting the null values.

**6. Find Books with Multiple Authors and Sort by Most Authors**

Find all books that have multiple authors and sort the results by the number of authors in descending order.

Answer :

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

To find books with multiple authors and sort them, you simply need to **filter** the collection for all books where the authorCount field is **greater than 1**.

After filtering, you **sort** those results by the authorCount field in **descending order** (highest to lowest), which places the books with the most authors at the top of the list.

**Lab – 4 Find the No. of persons**

We have the dataset given:

A screen shot of a computer screen

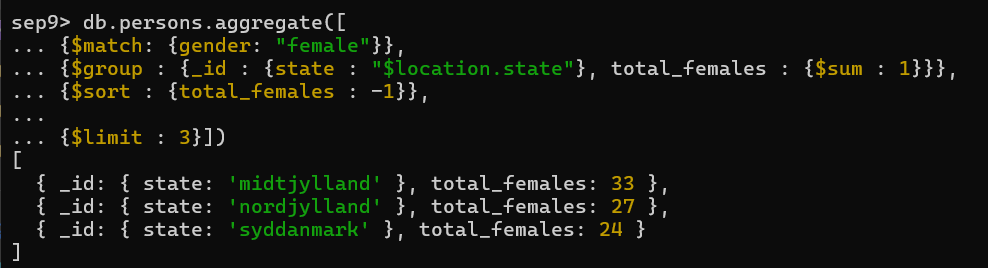
AI-generated content may be incorrect.

Dataset link:

Members.json : <https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/3/files/members.json>

After importing the dataset in mangoDB we have to answer the following question:

1. Find the No. of males and females:



**Conclusion:**

The query successfully identified the **top 3 states with the highest number of females** from the persons collection.

It worked by:

1. **Filtering** all documents to find only those where gender is "female".
2. **Grouping** these females by their location.state.
3. **Counting** the number of females in each state.
4. **Sorting** these states by their female count in descending order.
5. **Limiting** the output to the top 3.

The result shows that **'midtjylland'** has the most females (33), followed by **'nordjylland'** (27), and **'syddanmark'** (24).

**Lab Assignment 5**

**Dataset Link:**

* 1. **HDFS Commands :** [**https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/HDFS%20Commands.pdf**](https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/HDFS%20Commands.pdf)
  2. **Pig First Program :** [**https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/Pig\_First\_Program.zip**](https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/Pig_First_Program.zip)
  3. **Map Reduce :** [**https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/Programs%20for%20MapReduce.zip**](https://github.com/Abhi-VIT/Big-Data-and-nosql/blob/main/4/Hadoop/Programs%20for%20MapReduce.zip)

**Neo4j – Summary**

**This notebook explores graph-based data modeling and analysis using Neo4j, emphasizing:  
• Representation of entities and relationships through nodes, edges, and properties.  
• Use of the Cypher query language for expressive graph querying and pattern matching.  
• Visualization of connected data for insights into networks and relationships.  
It serves as a strong educational tool for:  
• Understanding graph databases and their structure,  
• Building and querying real-world relationship networks,  
• Applying graph theory concepts in data science and analytics.**

**MongoDB – Summary**

**This notebook demonstrates document-based data management and analysis using MongoDB, focusing on:  
• Flexible JSON/BSON document structures for semi-structured data.  
• CRUD operations, indexing, and aggregation pipelines for data manipulation.  
• Integration of MongoDB with analytical tools for efficient querying and visualization.  
It serves as a valuable educational resource for:  
• Learning NoSQL data modeling concepts,  
• Managing and analyzing unstructured or rapidly evolving data,  
• Applying MongoDB in modern web and data-driven applications.**

**Hadoop – Summary**

**This notebook introduces distributed data storage and processing using Hadoop, highlighting:  
• The use of HDFS for scalable and fault-tolerant data storage.  
• Implementation of MapReduce for parallel data processing.  
• Integration with the Hadoop ecosystem (Hive, Pig, and Spark) for extended analytics.  
It serves as an effective educational tool for:  
• Understanding big data architecture and workflow,  
• Exploring batch processing and large-scale computation,  
• Applying Hadoop concepts in data engineering and analytics projects.**

# 

# **HADOOP (CLOUDERA)**

## **Introduction To Cloudera**

In the era of big data, the upcoming laboratory experiment embarks on an introduction to Cloudera, a comprehensive platform designed to tackle the challenges of managing and processing vast datasets. Cloudera represents a pioneering solution, providing a unified framework that seamlessly integrates various open-source components such as Apache Hadoop, Apache Spark, and Apache Hive.

The experiment seeks to demystify the functionalities of Cloudera, emphasizing its role in facilitating scalable, distributed computing across clusters of commodity hardware. The platform's versatility makes it a cornerstone in the field of big data analytics, enabling organizations to harness the power of diverse tools within a unified ecosystem.

Our exploration will delve into the core components of Cloudera, unraveling its capacity to handle storage, processing, and analysis of massive datasets. As we navigate through the experiment, we aim to gain practical insights into Cloudera's user-friendly interface, its role in managing data workflows, and its ability to empower data scientists and engineers in extracting meaningful insights from large and complex datasets. Through this endeavor, we seek to pave the way for a comprehensive understanding of Cloudera's pivotal role in the ever-evolving landscape of big data technologies.

**Lab Assignment 6**

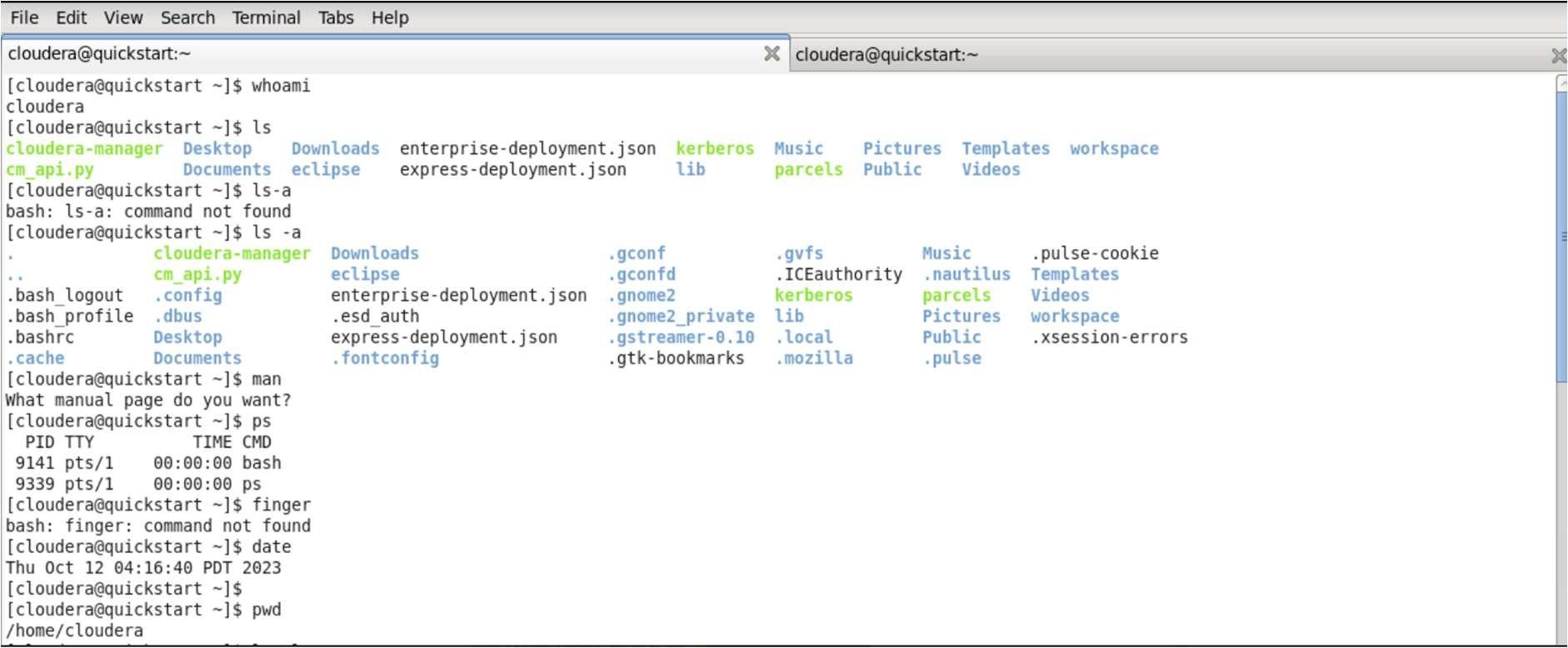
**Commands**:

**whoami**: This command prints the username of the current user. It's useful to check which user account you are currently using.

**ls -a**: The ls command is used to list files and directories. The -a option stands for "all," and it shows hidden files.

**man**: This command is used to display the manual pages for other commands.

**pwd**: This command stands for "print working directory," and it outputs the current working directory.



In Cloudera environments, the ls -l command is typically used in the command-line interface to list detailed information about files and directories in a specified directory. Here's a breakdown of the ls -l command and its output:

**ls**: This command is used to list files and directories.

-l: This option stands for "long format," and it provides detailed information about each file or directory, including permissions, owner, group, size, modification date, and the name of the file or directory.



The ls -ltr command is an extension of the ls -l command and adds the -t and -r options. Here's what each option does:

ls: List files and directories.

-l: Use the long format, displaying detailed information about each file.

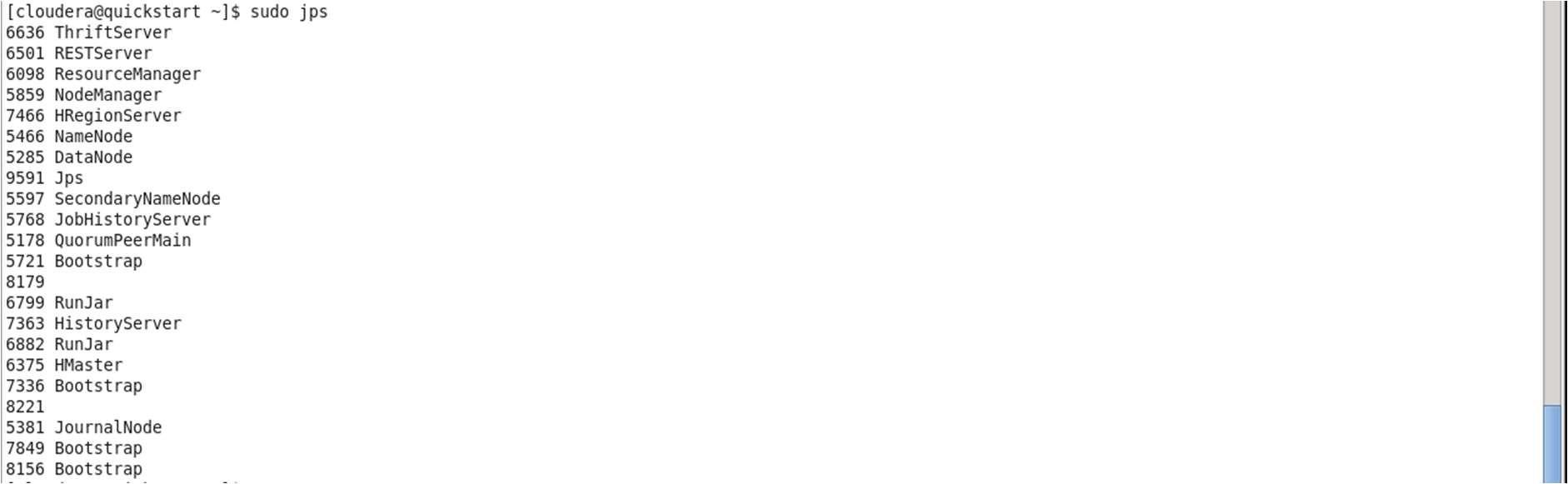
-t: Sort by modification time, showing the newest files or directories first.

-r: Reverse the order of the sort, so the oldest files or directories come first.

ls -ltr will list files and directories in the current directory in long format, sorted by modification time, with the newest items appearing at the bottom.

The sudo jps command is used to list the Java Virtual Machines (JVMs) that are running on a machine. The jps command is part of the Java Development Kit (JDK) and is commonly used to check which Java processes are currently running.

When you run sudo jps in a Cloudera environment, you'll typically see a list of Java processes associated with various Cloudera services. This can include processes for Hadoop daemons, HBase, Hive, Impala, and other components depending on your Cloudera distribution and the services you have configured.



In the culmination of this enlightening experiment introducing Cloudera, our exploration has unraveled the intricate capabilities of this pioneering platform in the era of big data. Cloudera's unified framework, seamlessly integrating Apache Hadoop, Apache Spark, and Apache Hive, emerged as a formidable solution designed to address the challenges inherent in managing and processing vast datasets.

Our journey through the core components of Cloudera delved into its capacity to handle storage, processing, and analysis of massive datasets. The platform's

versatility became evident, serving as a cornerstone in the field of big data analytics. As we navigated through the experiment, practical insights were gained into Cloudera's user-friendly interface, its pivotal role in managing data workflows, and its empowering potential for data scientists and engineers.

This exploration has paved the way for a comprehensive understanding of Cloudera's significance in the ever-evolving landscape of big data technologies. By demystifying its functionalities, the experiment not only equipped participants with practical skills but also highlighted Cloudera's role as a catalyst for extracting meaningful insights from large and complex datasets. This journey stands testament to Cloudera's transformative impact in empowering organizations to harness the full potential of diverse tools within a unified ecosystem, propelling them towards the forefront of innovation in big data analytics.

**Lab Assignment 7**

## **Introduction To MapReduce**

The next laboratory experiment embarks on an exploration of MapReduce, a fundamental programming model that has revolutionized the landscape of distributed computing and data processing. Developed to address the challenges posed by large-scale data sets, MapReduce provides a scalable and parallelized approach to processing vast amounts of information across distributed computing clusters.

At its core, MapReduce simplifies the complex task of data processing by breaking it into two distinct phases: the "Map" phase for parallel data processing, and the "Reduce" phase for aggregating and summarizing the results. This paradigm has become instrumental in enabling efficient computation on extensive datasets, making it a cornerstone in the development of big data applications.

Our experiment aims to unravel the intricacies of MapReduce, exploring its architecture, workflow, and its applicability in handling diverse data processing scenarios. By understanding the principles behind MapReduce, we seek to equip participants with the foundational knowledge necessary to harness the power of distributed computing for efficient and scalable data processing. Through this endeavor, we aspire to contribute to a deeper comprehension of MapReduce as a pivotal tool in the arsenal of big data technologies.

### **Commands:**

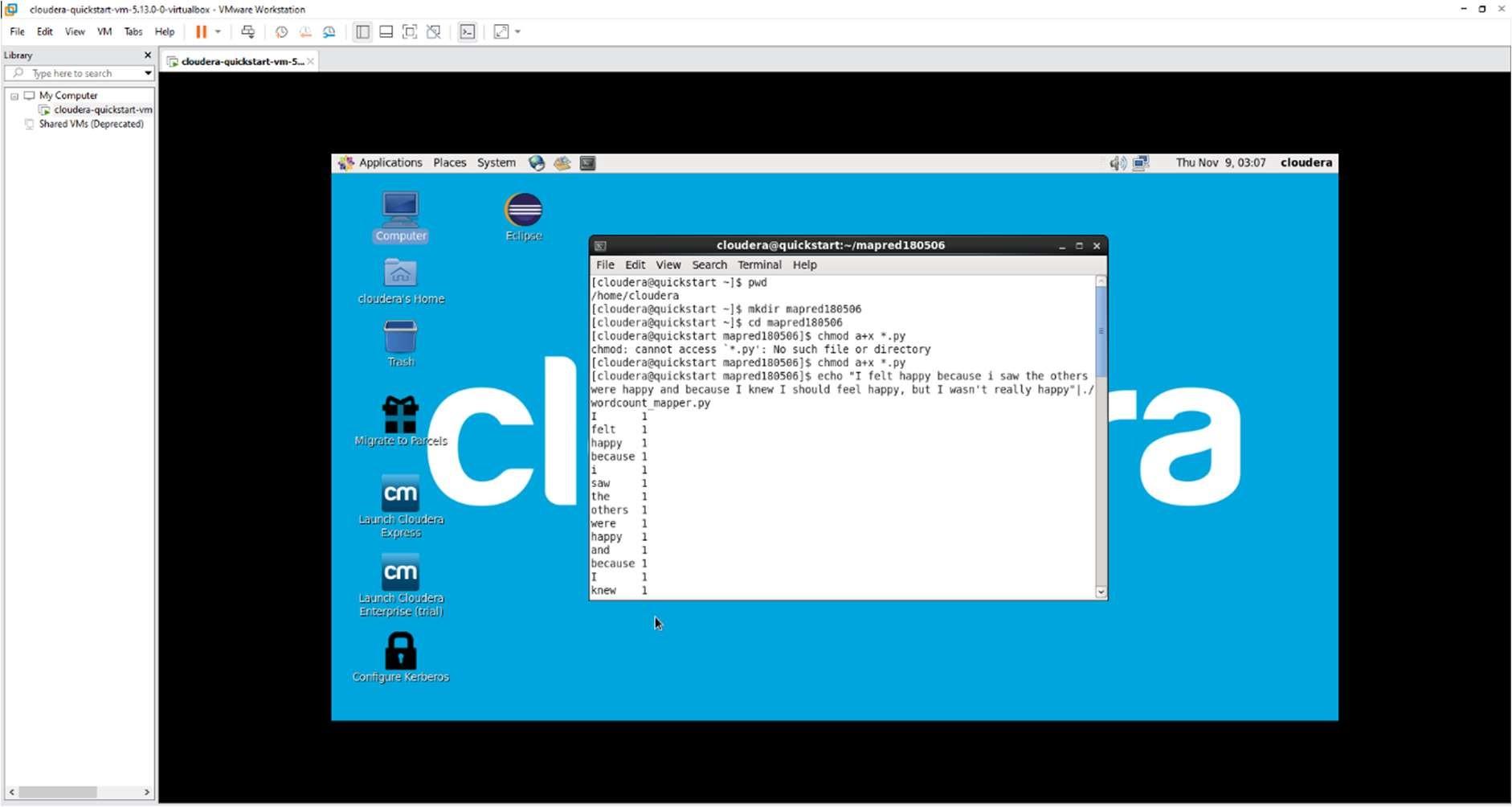
pwd: This command stands for "print working directory" and is used to display the current working directory.

mkdir mapred180506: This command creates a new directory named mapred180506.

cd mapred180506: This command changes the current working directory to mapred180506

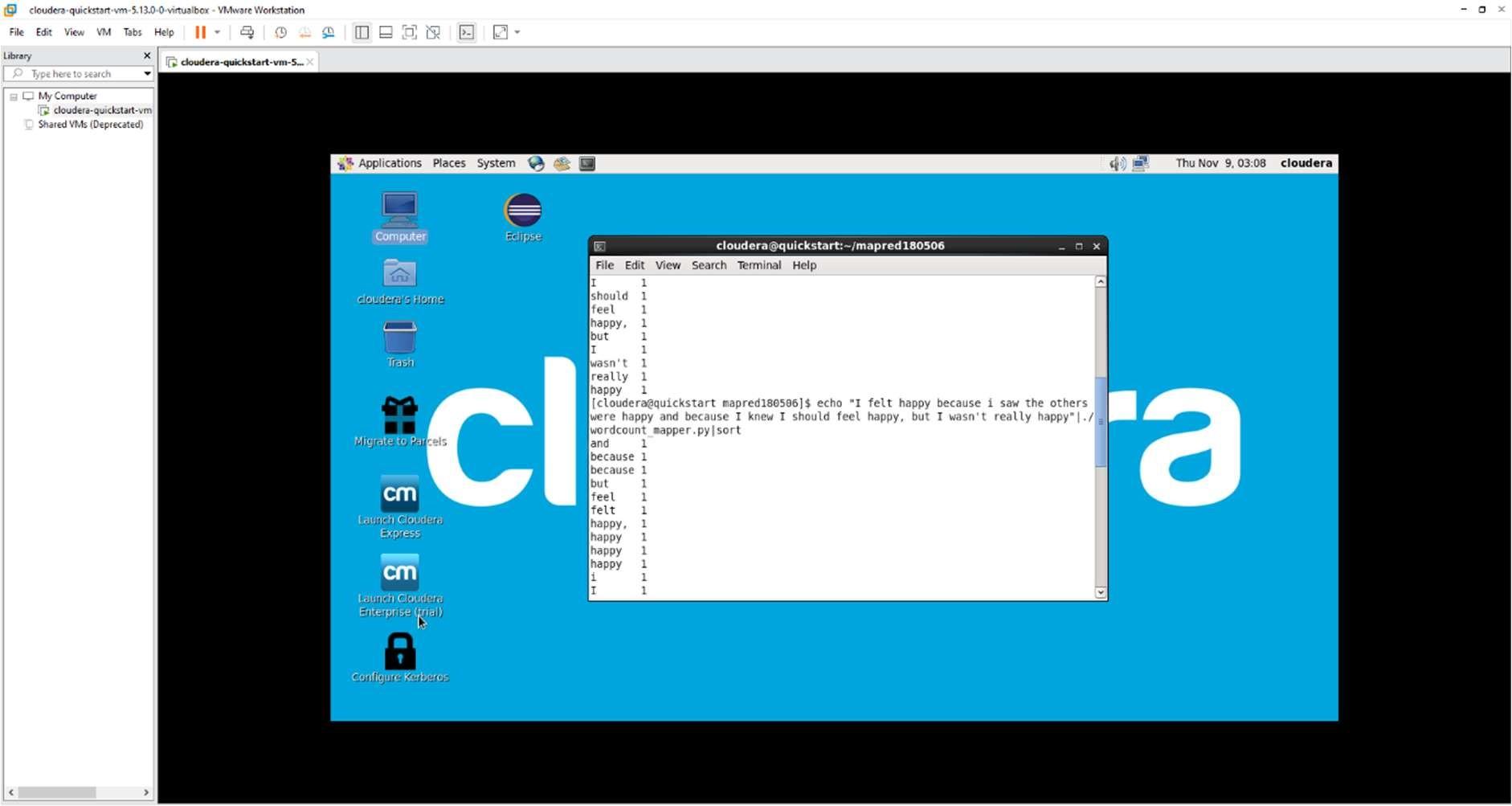
chmod a+x \*.py: This command grants execute permissions to all files with a .py extension in the current directory.

The Python script `wordcount\_mapper.py` reads lines from standard input, tokenizes words, and emits each word with a count of 1. This script is designed for a Hadoop MapReduce job to process text data, counting the occurrences of each word. The provided `echo` command pipes a test sentence into the script for demonstration.

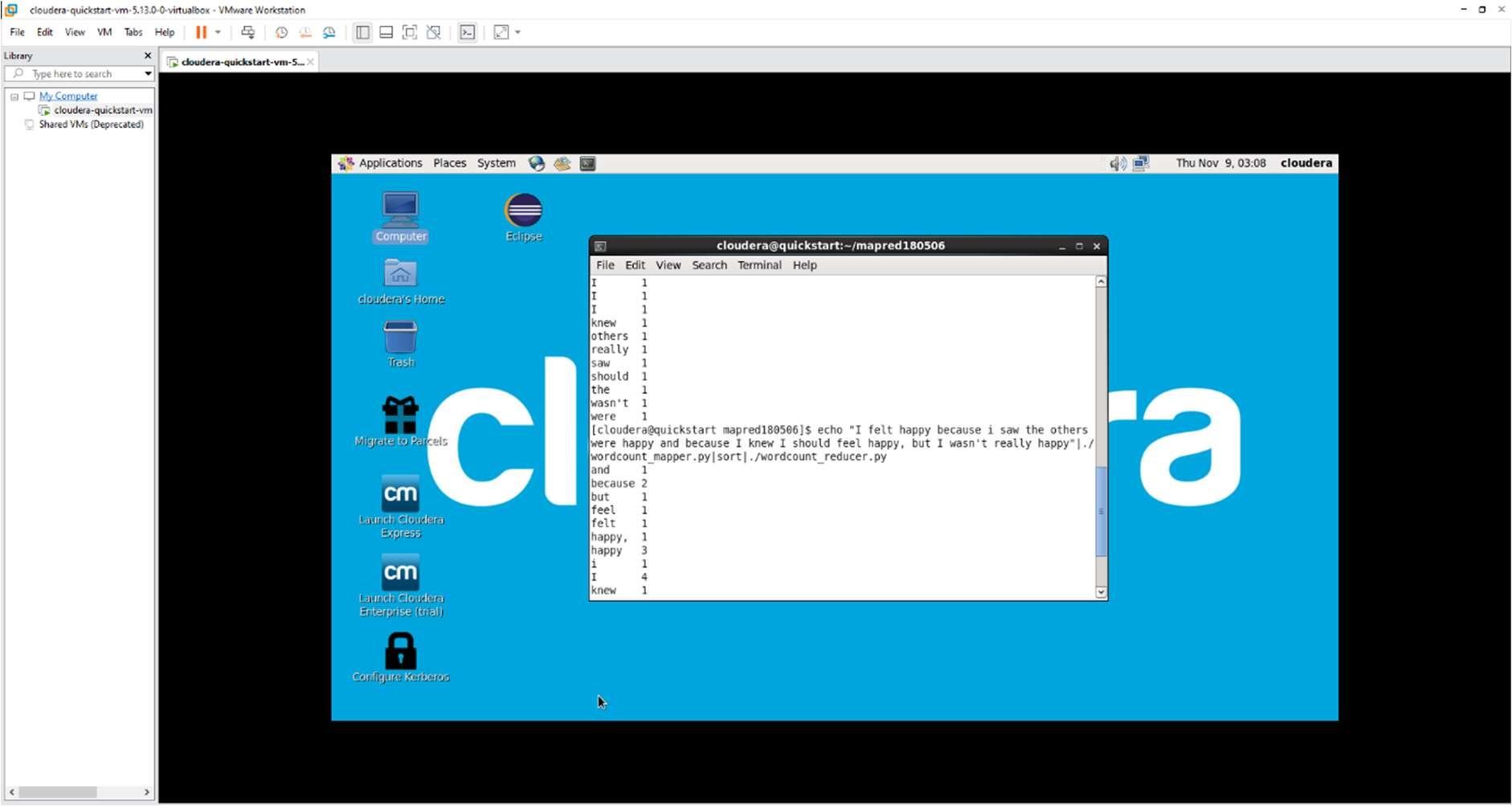


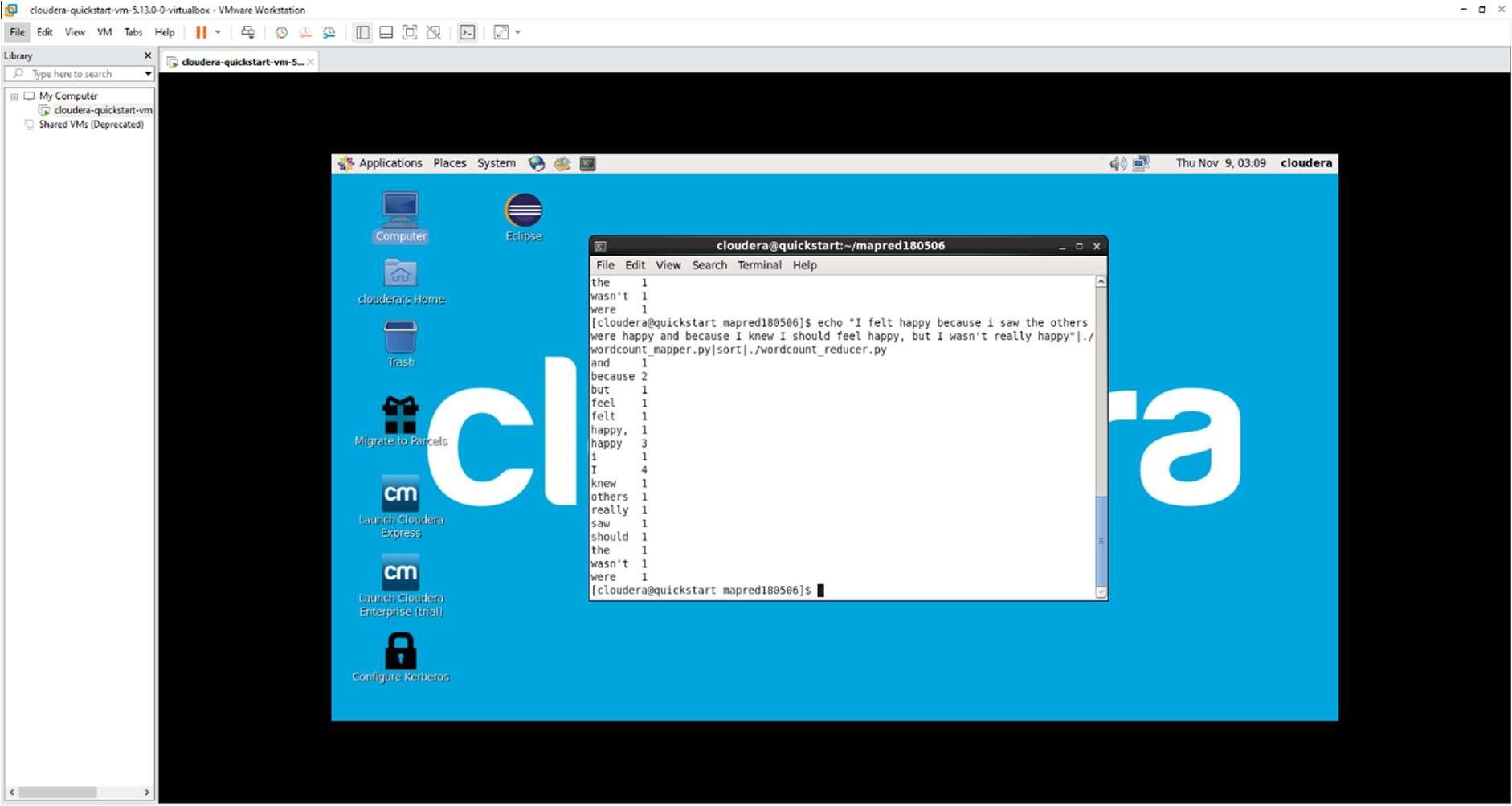
The command `echo "I felt happy because i saw the others were happy and because I knew I should feel happy, but I wasn't really happy" | ./wordcount\_mapper.py | sort` takes a sentence as input, processes it through the `wordcount\_mapper.py` script, which tokenizes words and emits counts, and then sorts the output alphabetically.

This sequence of commands is a common pattern in Hadoop MapReduce to preprocess and sort data for further analysis.



The command `echo "I felt happy because i saw the others were happy and because I knew I should feel happy, but I wasn't really happy" | ./wordcount\_mapper.py | sort |

./wordcount\_reducer.py` processes a sentence through a word count mapper (`wordcount\_mapper.py`), sorts the output, and then passes it to a word count reducer (`wordcount\_reducer.py`). This sequence emulates the MapReduce paradigm, where data is mapped, sorted, and reduced to calculate word frequencies.



As we conclude this exploration into the realms of MapReduce, our laboratory experiment has delved into the transformative world of distributed computing and data processing. MapReduce, a fundamental programming model, has been unveiled as a powerful tool that revolutionizes the landscape of handling large-scale datasets, providing a scalable and parallelized approach.

This experiment has elucidated the core principles of MapReduce, dissecting its architecture and workflow to unravel the intricacies of its parallel processing paradigm. The dual phases of "Map" and "Reduce" have been revealed as the linchpin in simplifying the complexities of data processing, enabling efficient computation on extensive datasets and standing as a cornerstone in the development of big data applications.

Our objective to equip participants with foundational knowledge has been met, paving the way for a deeper comprehension of MapReduce's role as a pivotal tool in the arsenal of big data technologies. By shedding light on its applicability in diverse data processing scenarios, this experiment contributes to the broader understanding of distributed computing, empowering individuals to harness the power of MapReduce for scalable and efficient data processing in the era of big data.

# **Introduction To PIG**

In the dynamic landscape of big data processing, the upcoming laboratory experiment introduces participants to Apache Pig, a high-level scripting platform built on top of Hadoop. Aimed at simplifying the complexities of data processing and analysis, Pig provides an abstraction layer that enables users to express data transformations using a scripting language known as Pig Latin.

The experiment explores the versatility of Pig as a data flow scripting language, offering an intuitive and expressive way to manipulate and analyze large datasets without the need for intricate Java programming. With its abstraction over MapReduce, Pig facilitates the creation of complex data workflows, making it an invaluable tool for data engineers and analysts working with Hadoop clusters.

Our exploration into Pig delves into its scripting capabilities, the creation of data processing pipelines, and its seamless integration with Hadoop ecosystems. By navigating through this experiment, participants will gain practical insights into how Pig streamlines the process of ETL (Extract, Transform, Load) and analytics tasks, contributing to a broader understanding of its role in the realm of big data processing and analysis.

### **Commands:**

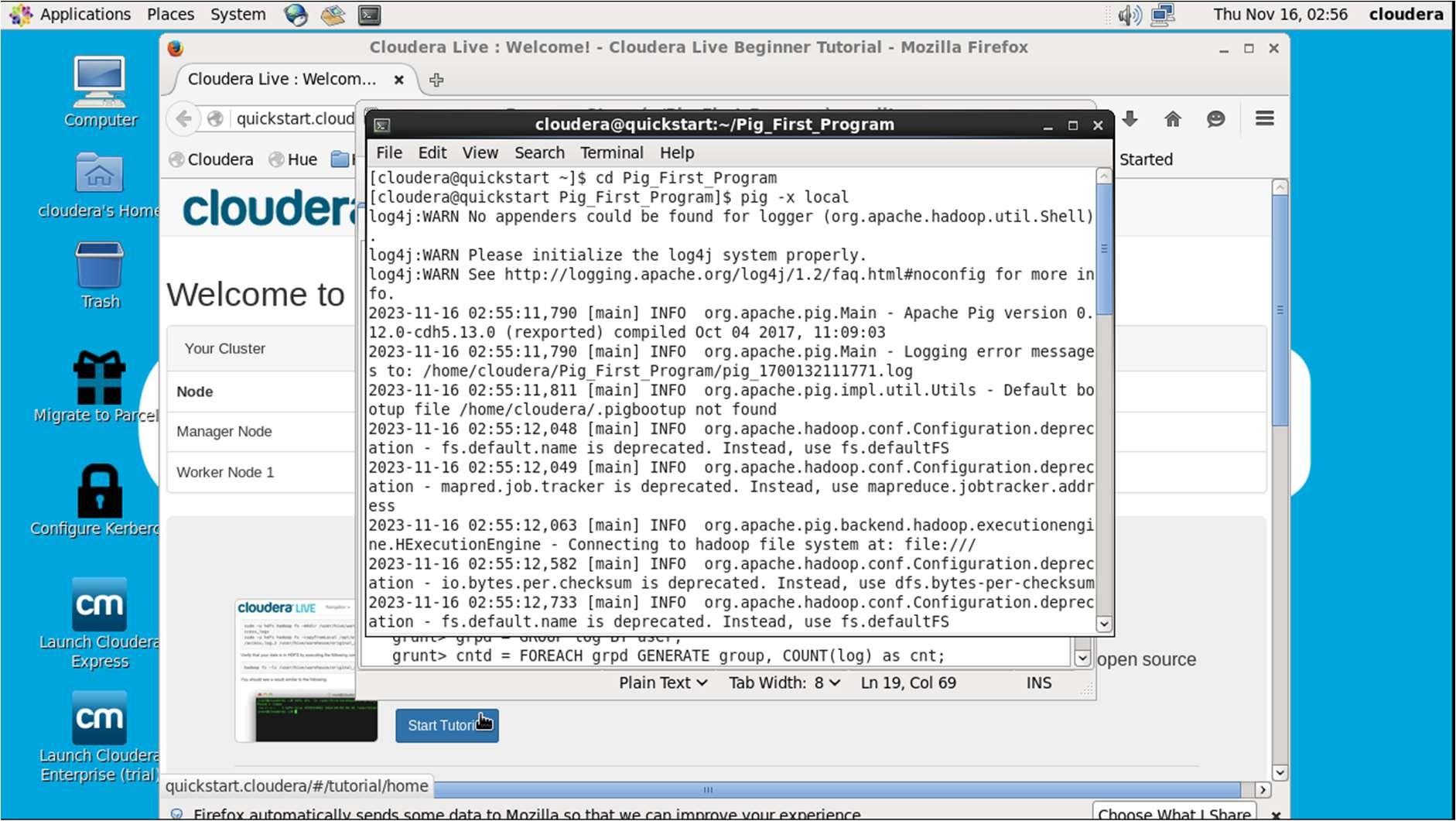
The command `pig -x local` is used to run Apache Pig in local mode. In this mode, Pig executes jobs on the local machine, without the need for a Hadoop cluster. This is particularly useful for testing and debugging Pig scripts on smaller datasets or when a full Hadoop cluster is not available.

Here's a brief breakdown of the command:

* `pig`: Initiates the Pig interpreter.
* `-x`: Specifies the execution mode.
* `local`: Indicates that Pig should run in local mode.

When you run Pig in local mode, it processes data on the local file system instead of distributing it across a Hadoop cluster. This can be advantageous for development and testing purposes before deploying Pig scripts to a larger-scale Hadoop environment.

Output:



grunt> log = LOAD 'excite-small.log' AS (user, timestamp, query);

This command loads data from the 'excite-small.log' file and defines a schema for the data with three fields: `user`, `timestamp`, and `query`.

grunt> grpd = GROUP log BY user;

Groups the data loaded in the previous step (`log`) by the `user` field.

grunt> cntd = FOREACH grpd GENERATE group, COUNT(log) as cnt;

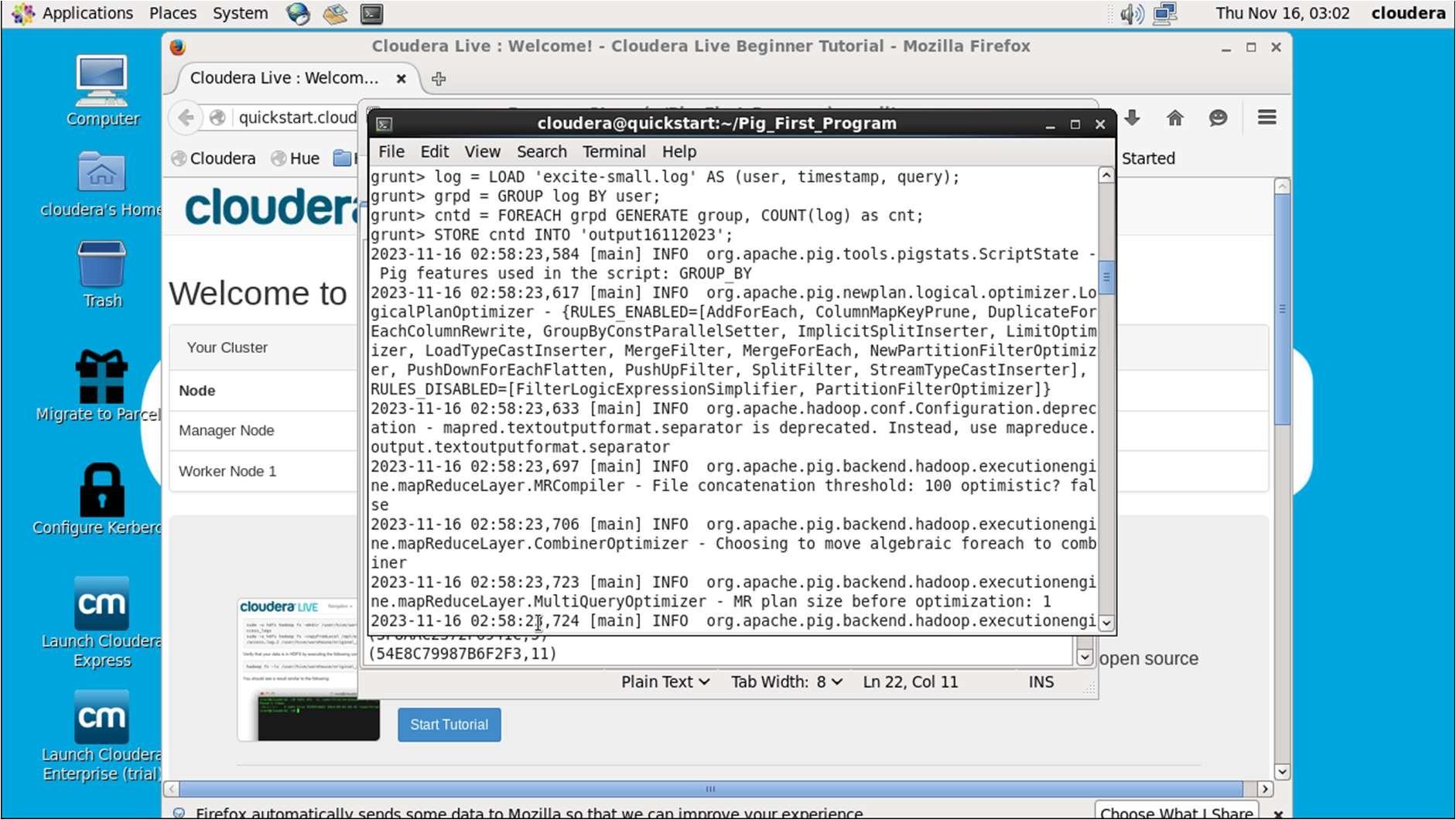
For each group created in the previous step (`grpd`), it generates a new dataset (`cntd`) with two fields: `group` (user) and `cnt` (count), representing the number of occurrences of each user in the original data.

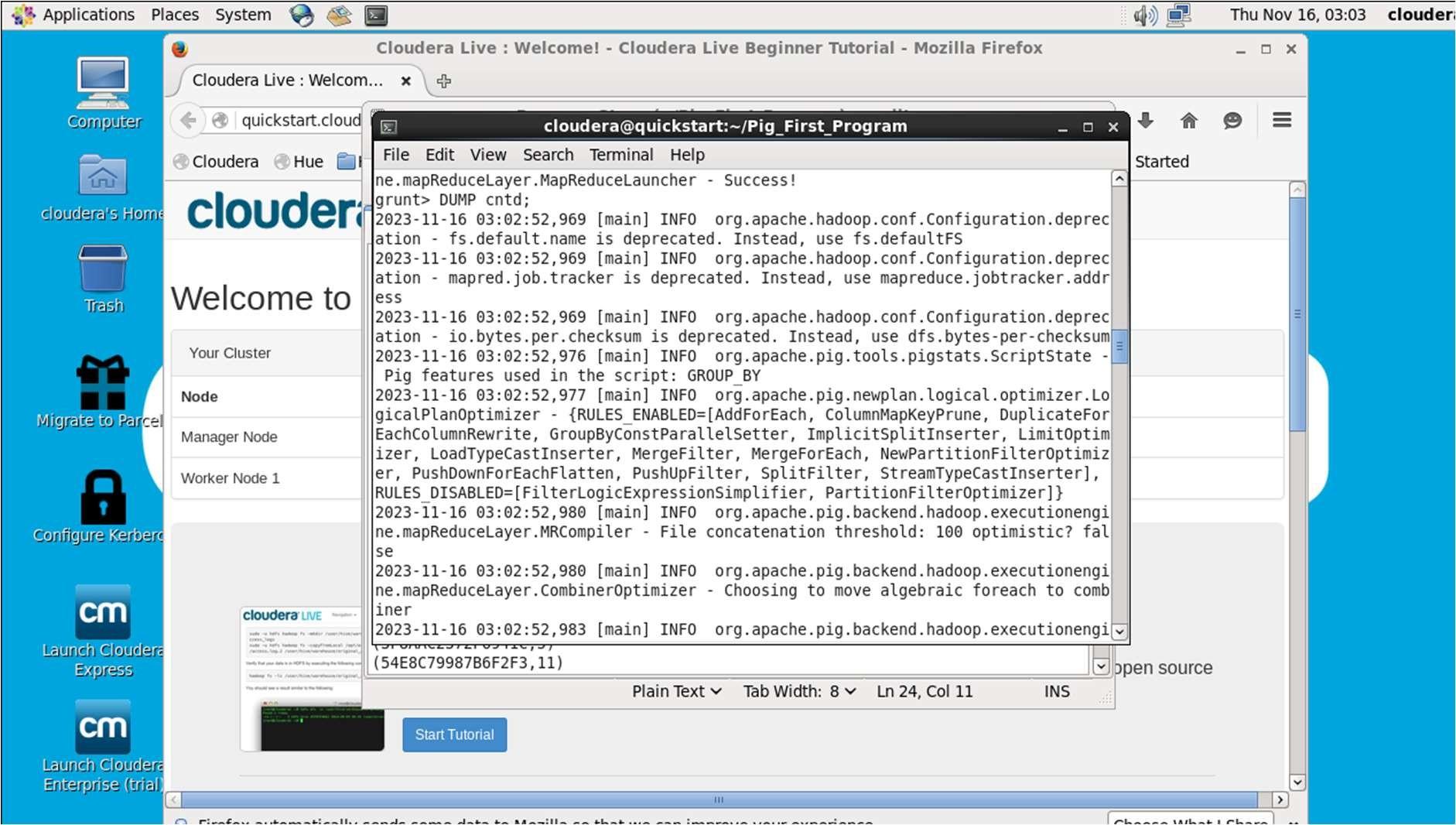
grunt> STORE cntd INTO 'output';

Stores the final result (`cntd`) into the 'output' directory or file.

This Pig script essentially counts the occurrences of each user in the 'excite- small.log' file and stores the results in the 'output' directory or file. Keep in mind that the 'excite-small.log' file should be in a format compatible with the specified schema, and the output directory should be writable.

OUTPUT :





grunt> fltrd = FILTER cntd BY cnt > 3;

This command filters the dataset `cntd` to include only those records where the count (`cnt`) is greater than 3.

grunt> STORE fltrd INTO 'output1';

Stores the filtered results (`fltrd`) in the 'output1' directory or file.

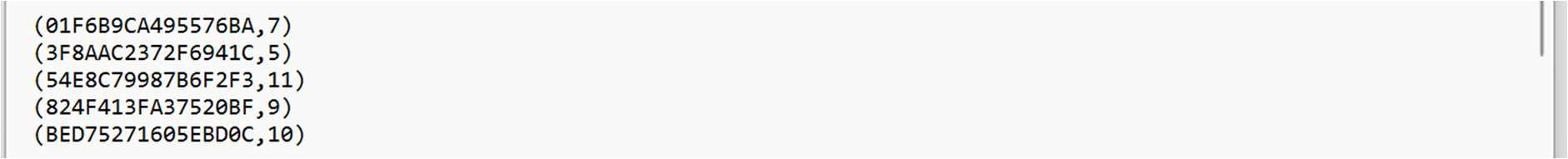
grunt> DUMP fltrd;

This command outputs the contents of the filtered dataset (`fltrd`) to the console. It's essentially a way to inspect the filtered results in the Pig interactive shell.

This script filters the previously counted dataset (`cntd`) to include only those users who appear more than three times in the original data. The filtered results are both stored in the 'output1' directory or file and displayed on the console using the

`DUMP` command.

OUTPUT :



grunt> srtd = ORDER fltrd BY cnt desc;

This command sorts the filtered dataset `fltrd` based on the `cnt` (count) field in descending order, creating a new dataset `srtd`.

grunt> STORE srtd INTO 'output2';

Stores the sorted results (`srtd`) in the 'output2' directory or file.

grunt> DUMP srtd;

Outputs the contents of the sorted dataset (`srtd`) to the console using the `DUMP` command.

This Pig script filters the original dataset (`cntd`) to include only users with counts greater than 3, then sorts these filtered results based on the count in descending order. The sorted results are both stored in the 'output2' directory or file and displayed on the console. This sequence of operations allows for meaningful analysis of the data, focusing on users with higher counts.

OUTPUT :



In the culmination of this immersive exploration into Apache Pig, our laboratory experiment has unfolded the layers of its significance in the dynamic landscape of big data processing. The experiment introduced participants to Pig as a high-level scripting platform, offering an abstraction layer designed to simplify the intricacies of data processing and analysis within the Hadoop ecosystem.

By delving into the capabilities of Pig as a data flow scripting language, participants gained a profound understanding of its intuitive and expressive nature. The experiment underscored Pig's role in mitigating the complexities associated with traditional Java programming, allowing data engineers and analysts to effortlessly express data transformations using Pig Latin.

The exploration extended to Pig's seamless integration with Hadoop ecosystems, illuminating its prowess in creating intricate data processing pipelines. As participants navigated through the experiment, practical insights were gained into Pig's role in streamlining ETL (Extract, Transform, Load) processes and analytics tasks. The experiment, therefore, significantly contributed to a broader comprehension of Pig's pivotal role in the realm of big data processing and analysis, showcasing its value as an indispensable tool for simplifying and enhancing data workflows within Hadoop clusters.

## **Introduction To Hbase**

In the ever-expanding landscape of big data storage and retrieval, the upcoming laboratory experiment immerses participants in the world of Apache HBase—an open- source, distributed, and scalable NoSQL database designed to handle vast amounts of sparse data. HBase operates atop the Hadoop Distributed File System (HDFS), leveraging its distributed and fault-tolerant architecture to provide efficient storage and retrieval capabilities for large-scale datasets.

This experiment delves into the fundamentals of HBase, exploring its column-family- oriented data model and its seamless integration with the Apache Hadoop ecosystem. Participants will gain hands-on experience in creating, manipulating, and querying data stored in HBase tables, as well as understanding the nuances of schema design in a NoSQL context.

Our exploration will navigate through the core concepts of HBase, including key-value stores, row keys, column families, and the underlying architecture that empowers real- time data access. By the end of this experiment, participants will be equipped with the foundational knowledge to harness the power of HBase for scalable and flexible data storage in scenarios demanding high throughput and low-latency access. The journey through this introduction to HBase promises to unlock insights into the realm of NoSQL databases and their pivotal role in modern big data architectures.

### **Command:**

[cloudera@quickstart ~]$ hbase shell

This starts the HBase shell, providing an interactive environment for executing HBase commands.

hbase(main):001:0> status

The `status` command displays the current status of the HBase cluster, showing the number of servers, dead servers, and average load.

hbase(main):004:0> help 'status'

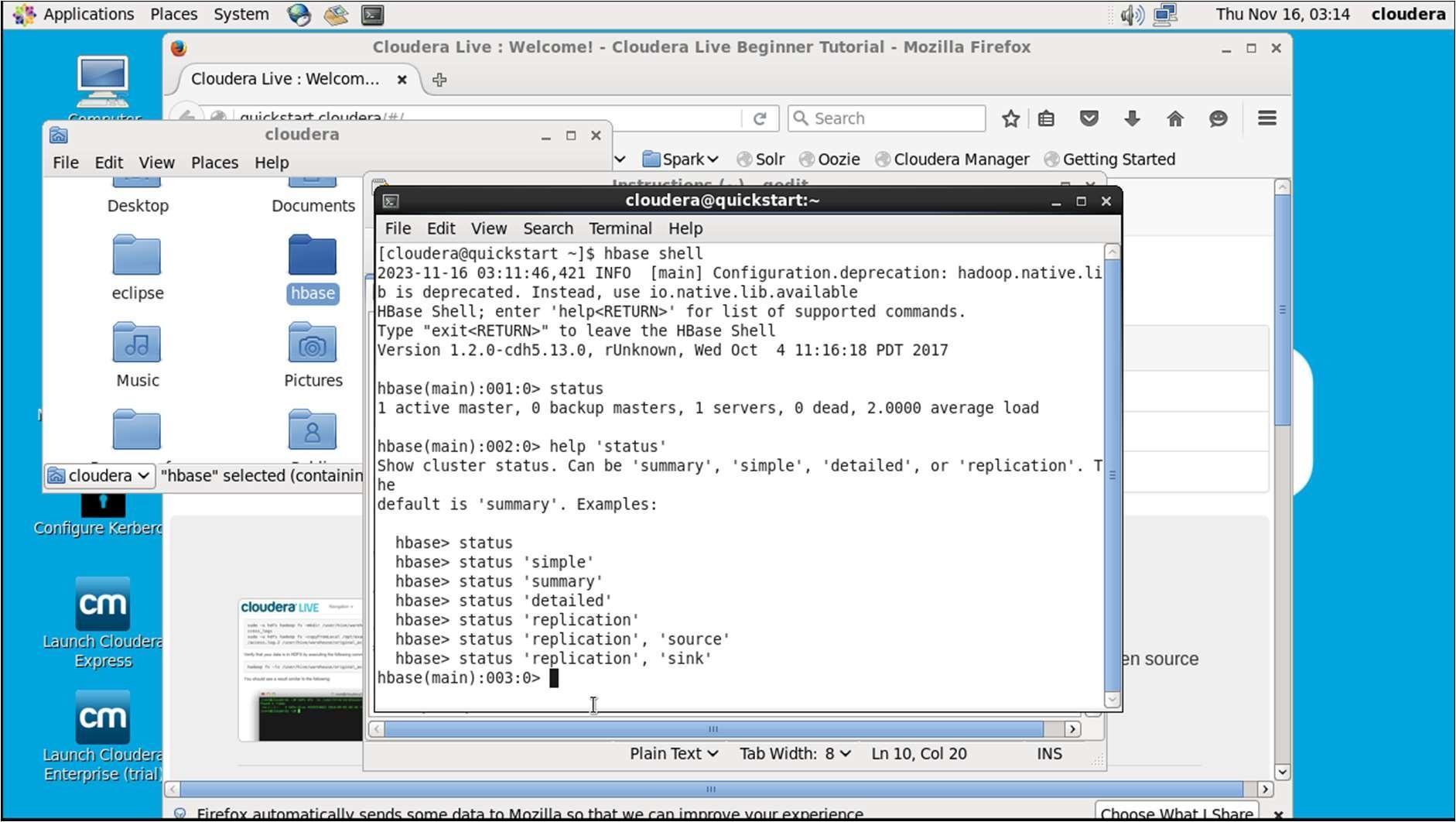
Show cluster status. Can be 'summary', 'simple', 'detailed', or 'replication'. The default is 'summary'.

The `help` command is used to get information about the usage and options of a specific HBase command (`status` in this example).

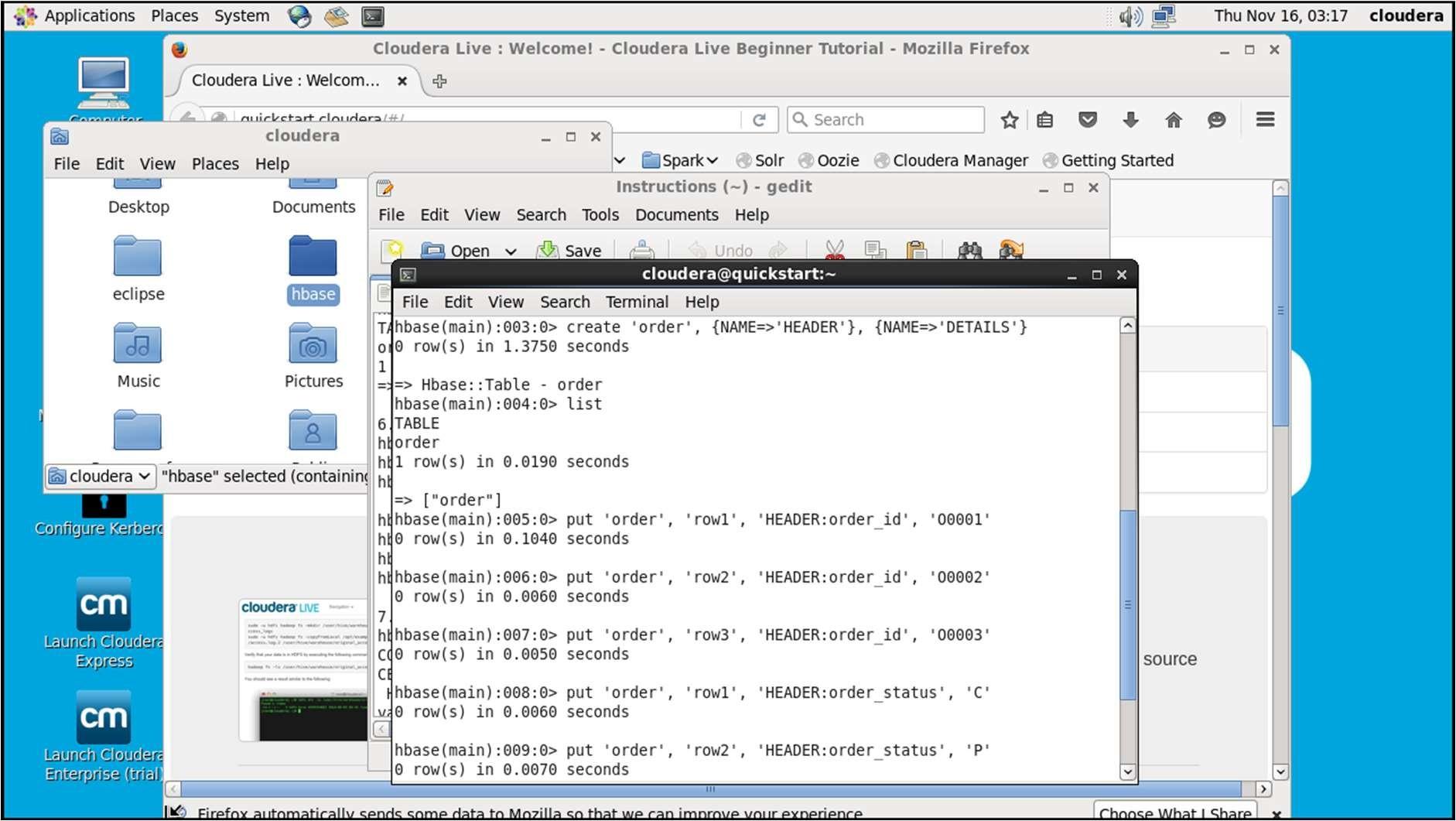
hbase(main):007:0> create 'order', {NAME=>'HEADER'}, {NAME=>'DETAILS'}

This command creates an HBase table named 'order' with two column families: 'HEADER' and 'DETAILS'.

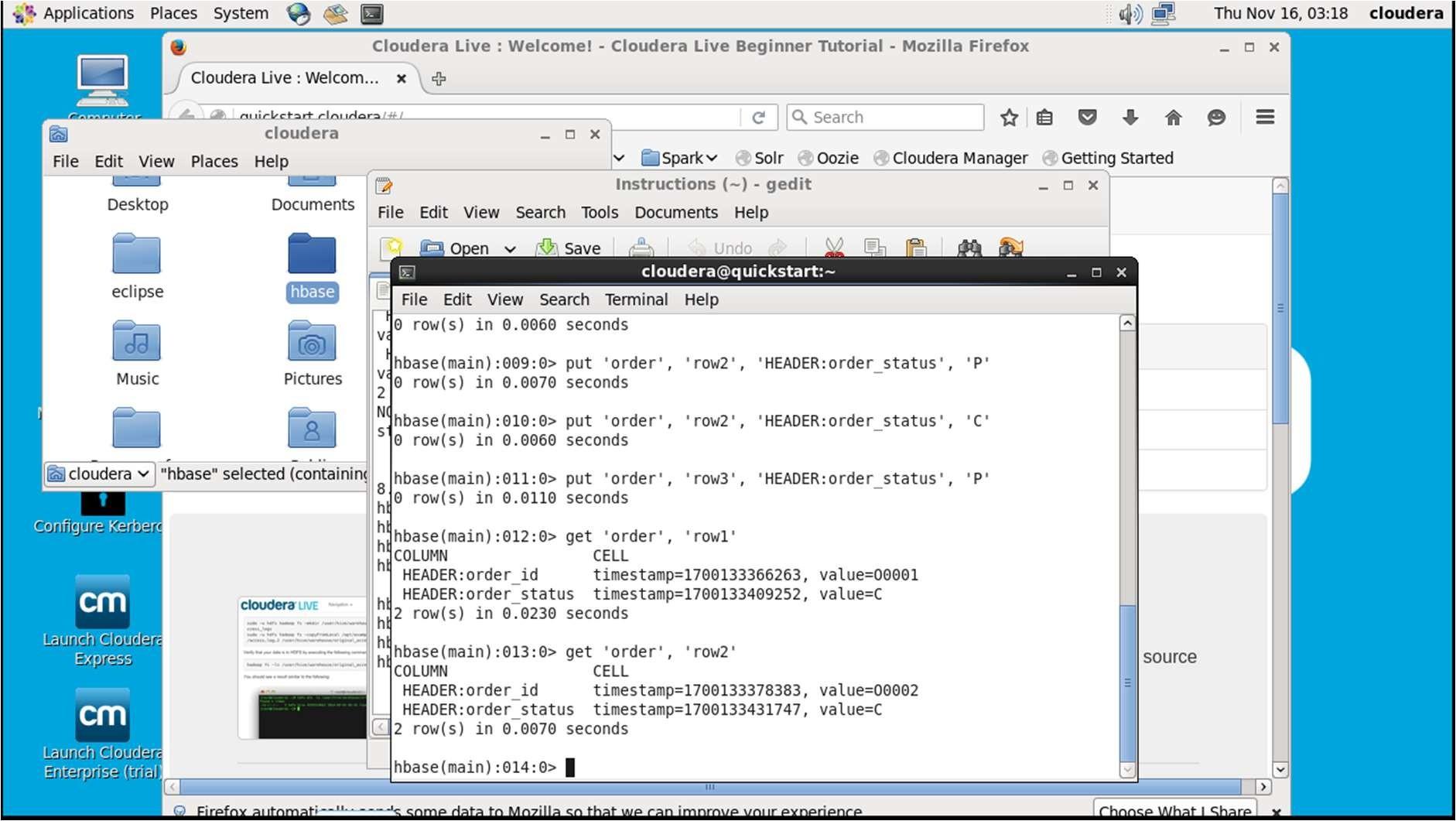
These commands demonstrate basic interactions with HBase, including checking the cluster status, getting help for a command, and creating an HBase table with specified column families. Make sure to adjust the table and column family names based on your specific use case.



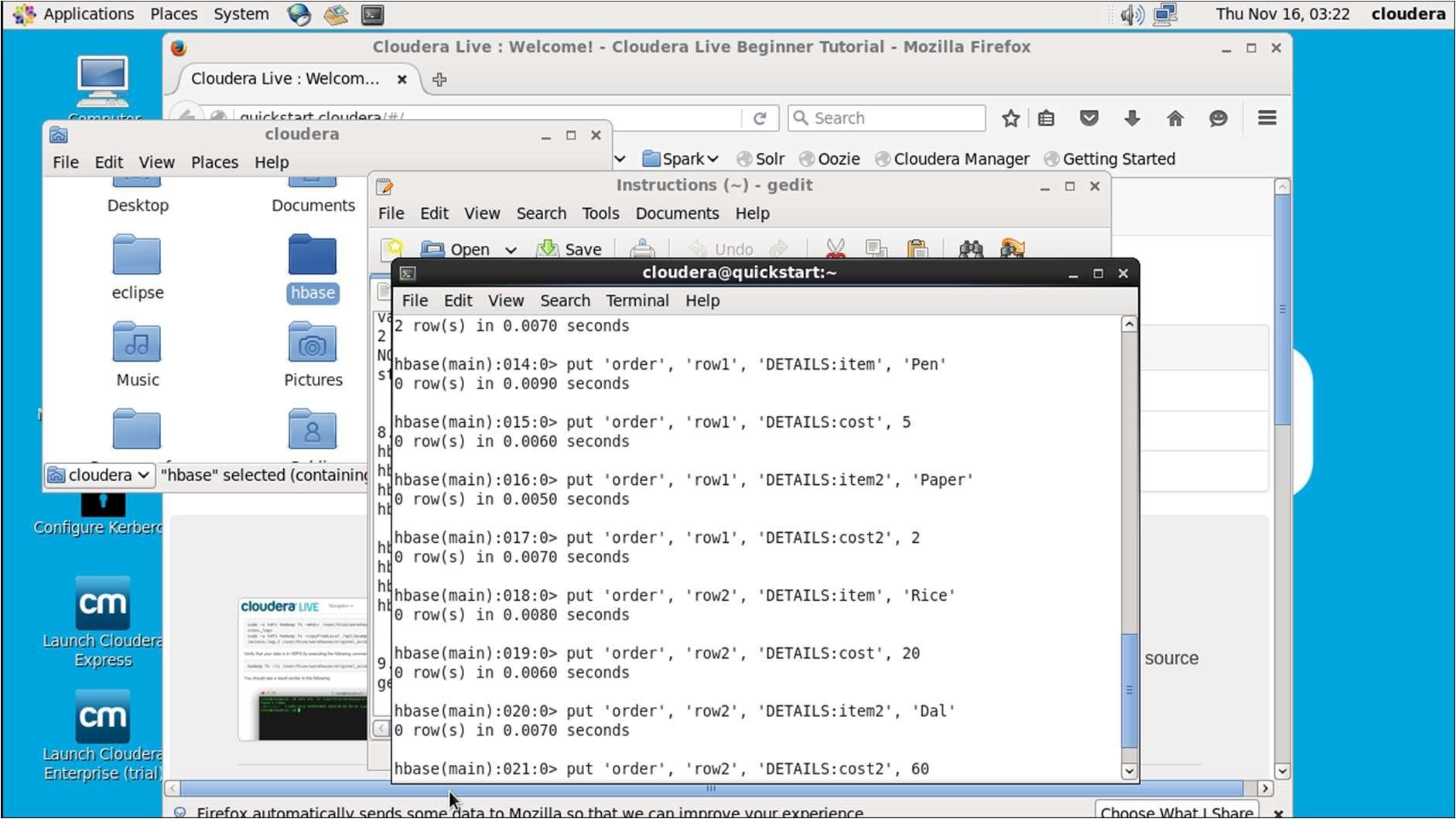
Inserting values into the database:



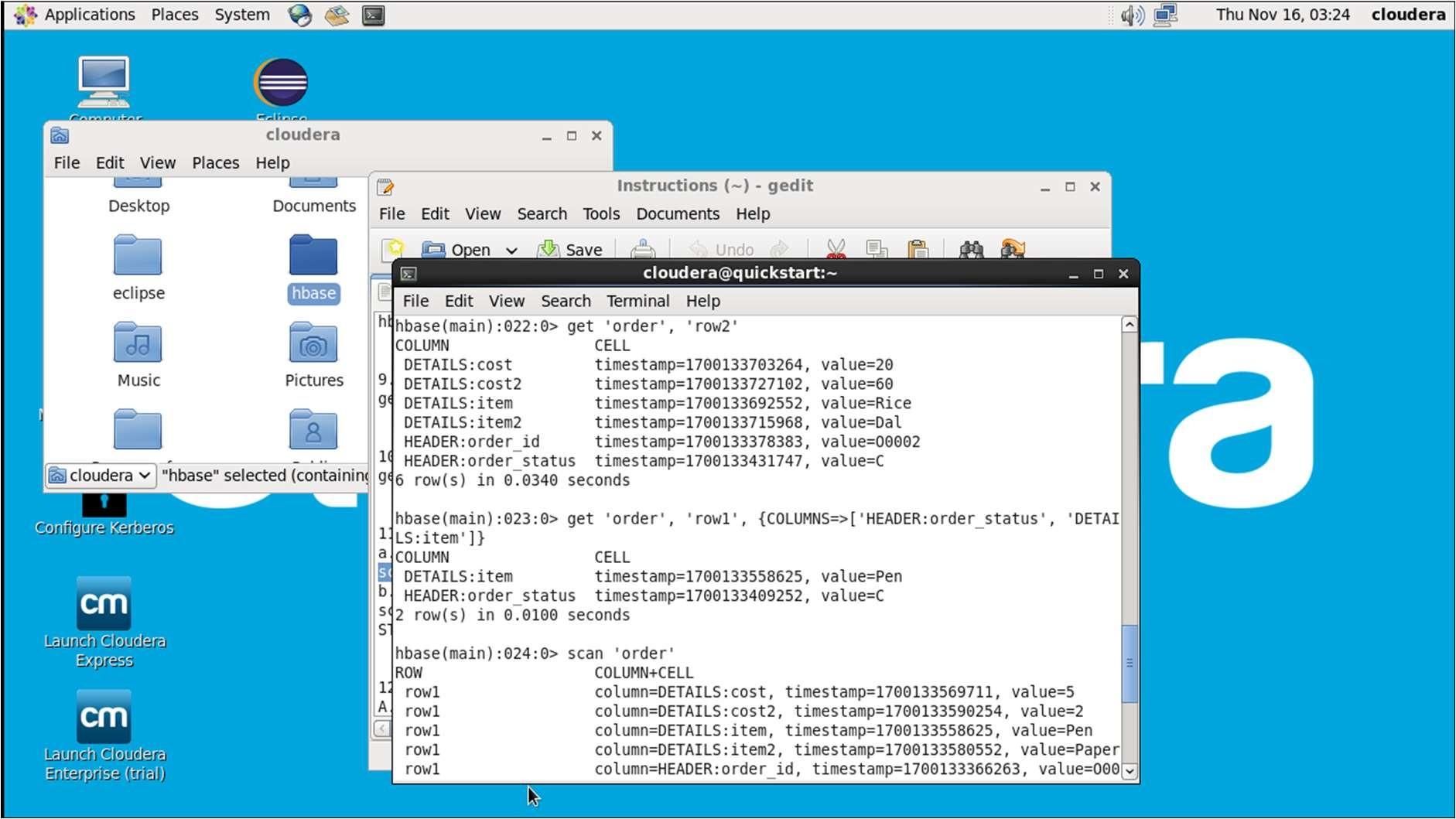
The command get 'order', 'row1' in HBase retrieves the data associated with the specified row key 'row1' from the 'order' table. This assumes that there is a row with the key 'row1' in the 'order' table.



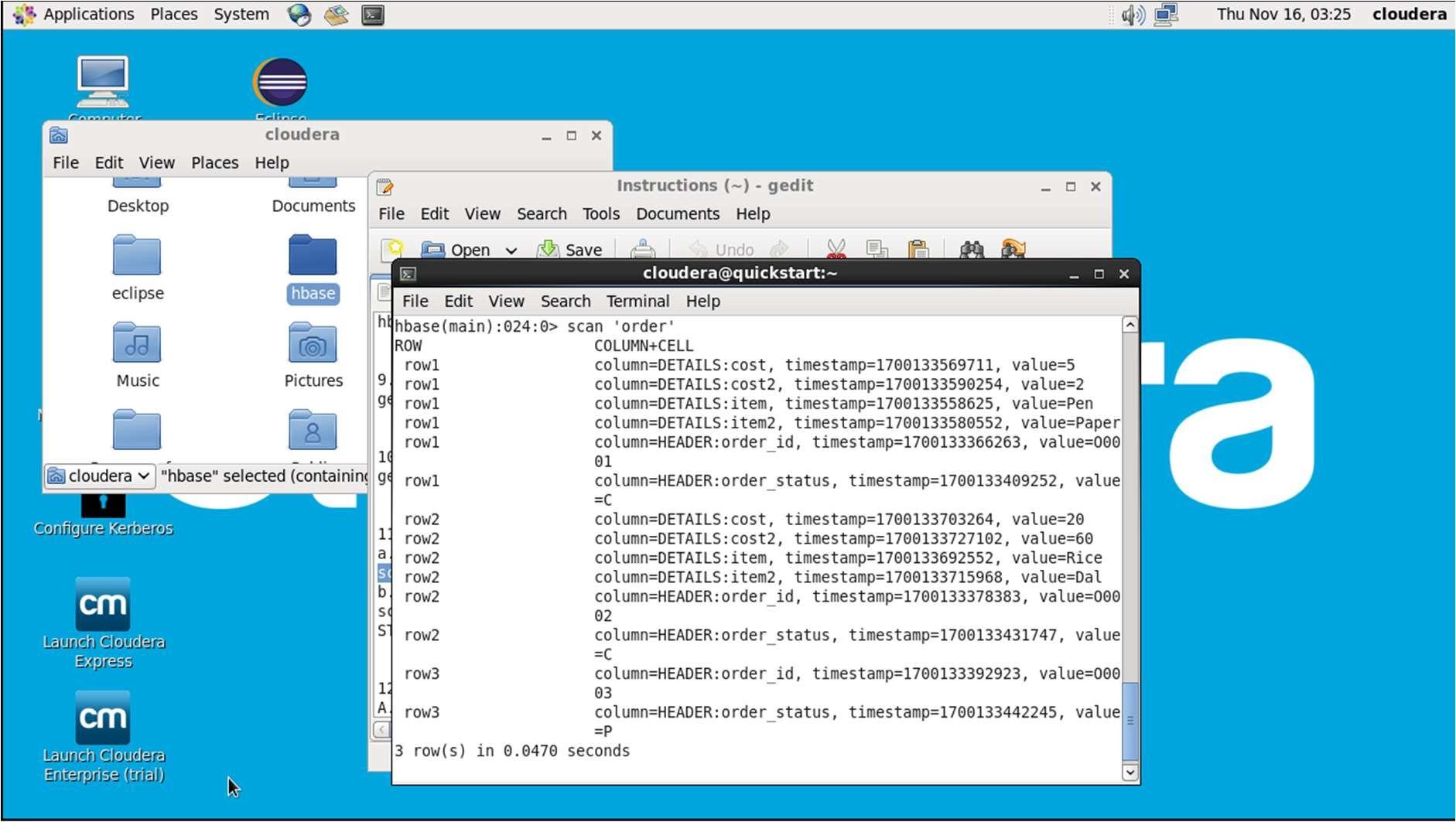
Put some data in second column family details:



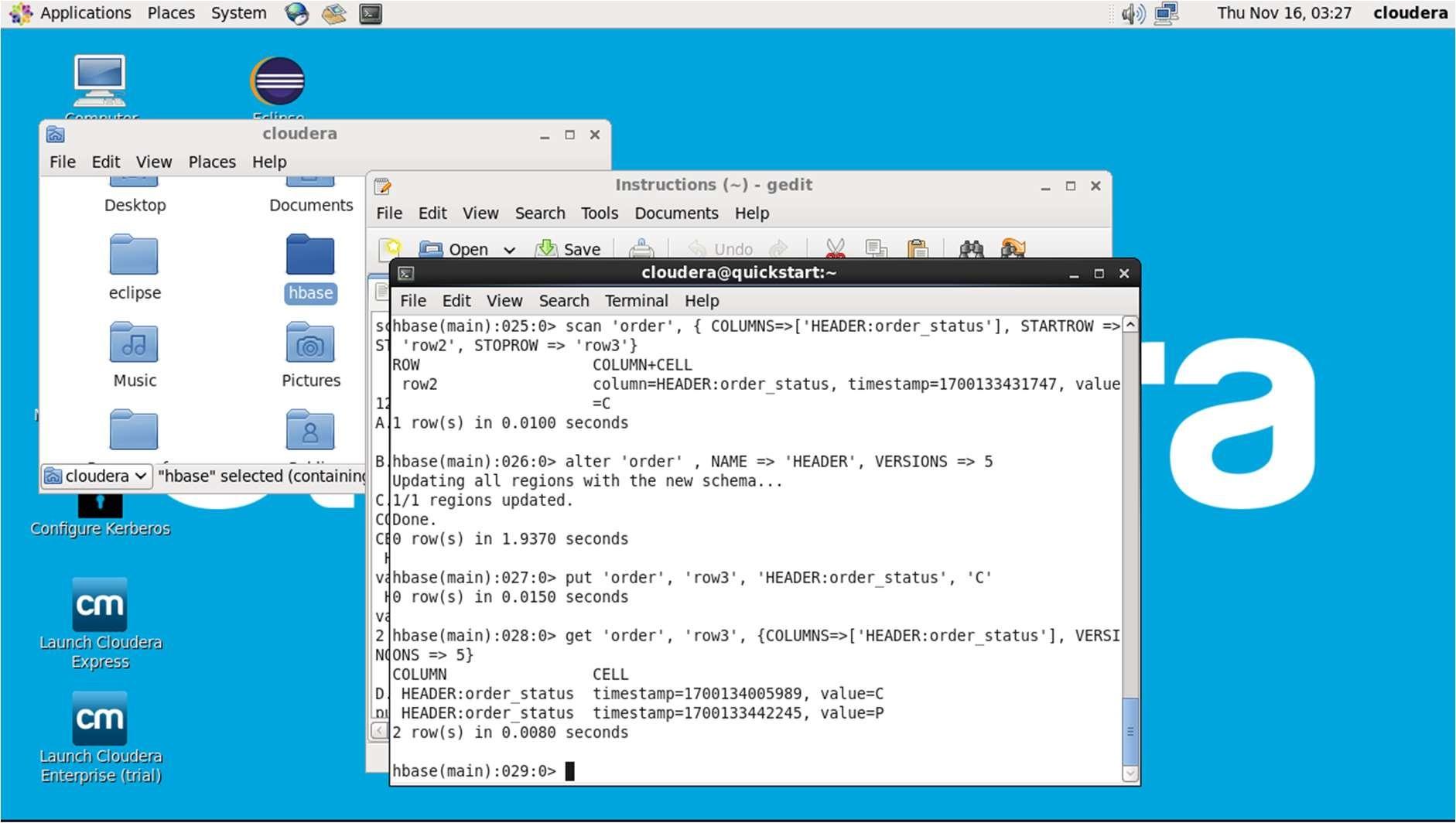
To view selective data



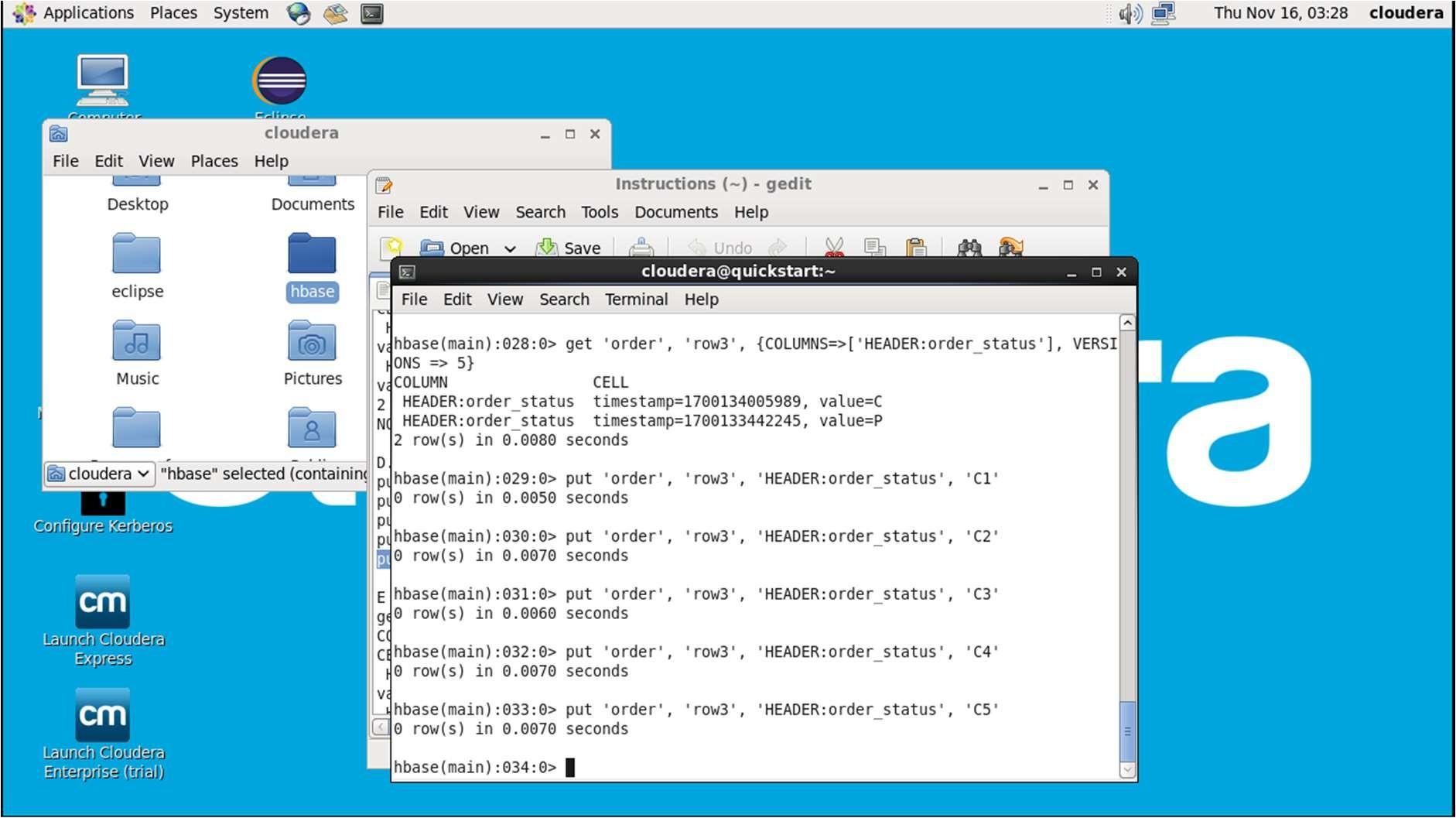
Scan entire table:



Scan with condition and Enable Versioning for column family 'HEADER' :

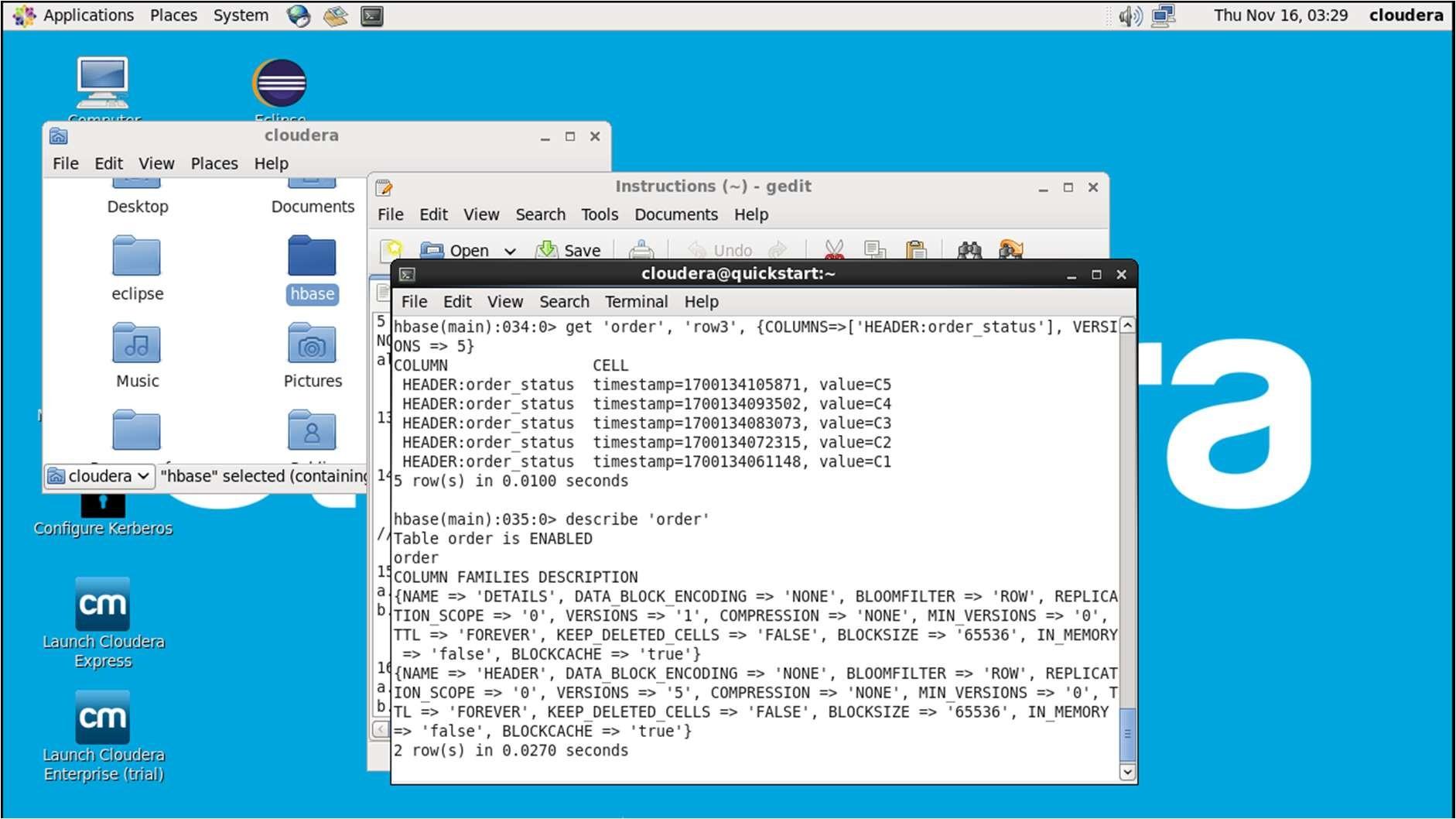


Inserting Values to the database:



The `describe` command in HBase is used to get information about a table's structure, including its column families and their associated qualifiers. When you run

`describe 'order'`, you'll get details about the 'order' table.



count 'order'

This command will count the number of rows in the 'order' table.

delete 'order', 'row2', 'HEADER:order\_status'

This command removes the value in the 'order\_status' column of the 'HEADER' family for the specified row ('row2') in the 'order' table.

deleteall 'order', 'row1'

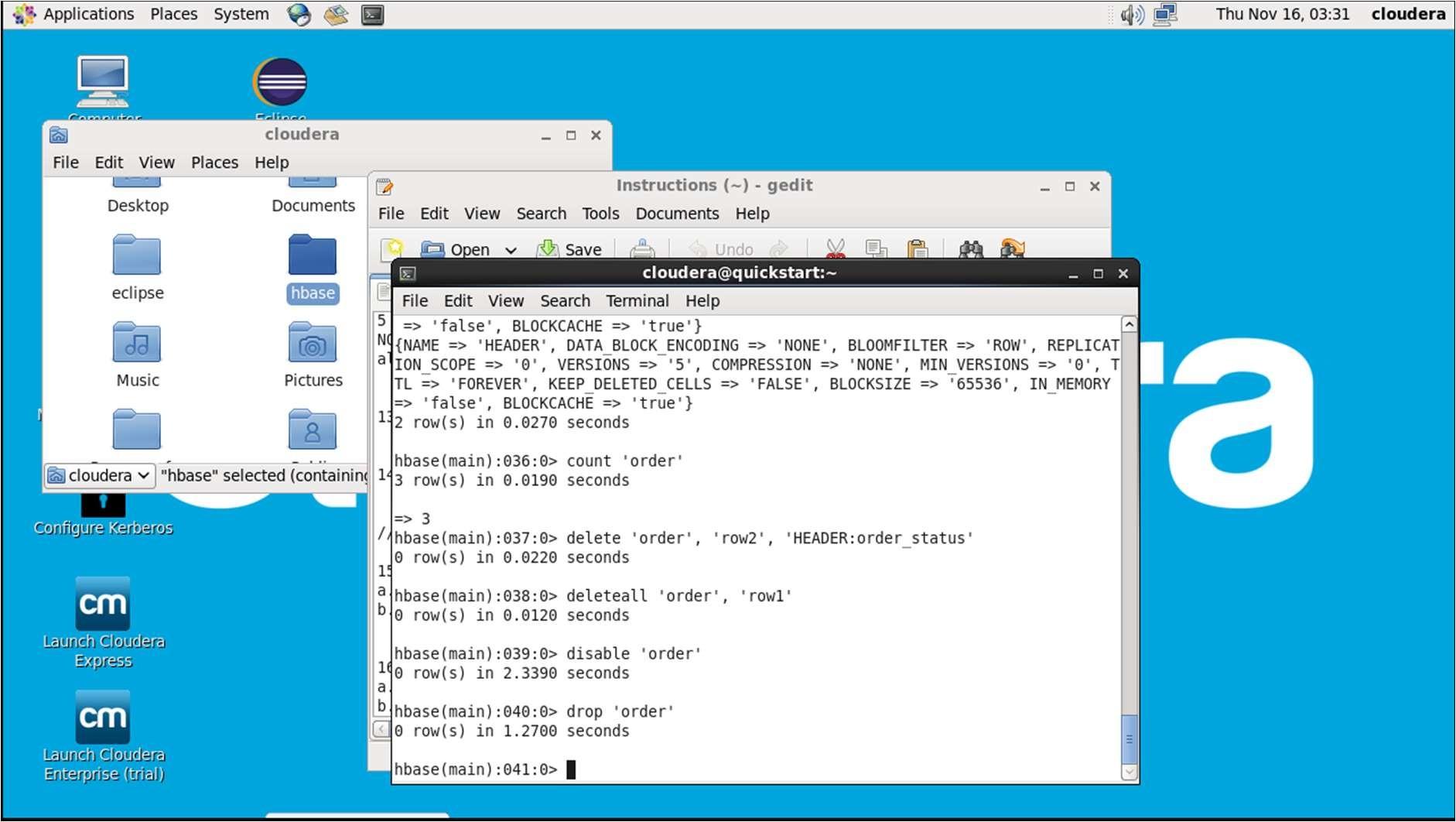
This command deletes all columns and families associated with the specified row ('row1') in the 'order' table.

disable 'order'

This command disables (takes offline) the 'order' table, making it unavailable for modifications.

drop 'order'

This command permanently deletes the 'order' table, including all data and metadata associated with it. Be cautious when using the `drop` command as it is irreversible.



In the culmination of this laboratory experiment exploring Apache HBase, participants have embarked on a journey through the intricacies of a distributed, NoSQL database designed for scalability and real-time data access. The experiment provided hands-on exposure to HBase's column-family-oriented data model,

leveraging its seamless integration with the Hadoop ecosystem for efficient storage and retrieval of vast datasets.

Throughout the exploration, participants engaged in creating, manipulating, and querying data within HBase tables, gaining practical insights into the nuances of schema design and the fundamental principles governing key-value stores. The experiment underscored HBase's significance in scenarios demanding high throughput and low-latency access, showcasing its potential in real-time data applications.

As we navigated the landscape of HBase, this experiment contributed to a deeper understanding of NoSQL databases, emphasizing HBase's role in modern big data architectures. Armed with newfound knowledge, participants are better equipped to harness the advantages of HBase for scalable and flexible data storage, propelling them into the forefront of contemporary data management practices.

## **CONCLUSION**

Throughout the series of experiments conducted in this lab, we have traversed the diverse landscapes of data management, processing, and storage within the realms of graph databases, distributed computing, and NoSQL technologies. The exploration of Neo4j unveiled its prowess in modeling complex relationships, offering valuable insights into the interconnected nature of data. The foray into Cloudera and MapReduce delved into the world of big data, providing a comprehensive understanding of scalable distributed computing. The introduction to Pig scripting showcased a simplified approach to data processing within the Hadoop ecosystem.

The latter part of our experiments shifted the focus to NoSQL databases, with an exploration of Apache HBase—a distributed, scalable solution for sparse data storage. From graph databases to distributed computing and NoSQL storage, each experiment contributed to a holistic comprehension of cutting-edge technologies in the data management landscape.

These experiments not only equipped participants with practical skills in utilizing tools such as Neo4j, Cloudera, MapReduce, Pig, and HBase but also fostered a deeper understanding of the underlying principles governing each technology. The ability to model, process, and analyze data within these diverse contexts is paramount in the ever-evolving field of data science and big data analytics.

As we conclude this lab report, the amalgamation of these experiments serves as a gateway to a nuanced comprehension of contemporary data management solutions. From intricate graph representations to scalable distributed computing and flexible NoSQL storage, the knowledge gained is a testament to the versatility and significance of these technologies in addressing the multifaceted challenges of modern data ecosystems. The journey through this lab has not only expanded our practical toolkit but has also ignited a curiosity to explore, experiment, and innovate in the dynamic field of data science and analytics.