Index

Practical No. 1

Aim: Perform DDL, DML and TCL queries in any RDBMS.

Explanation:
DDL in conjunction with DML and TCL can help manage and manipulate data and database structure in an RDBMS.
Data Definition Language: The commands of the DDL are used to define, change, or delete database structures and include commands that manipulate the table, index, and schema structures. They directly affect that which defines the database design.
Features

- Accordingly, the operations would be on the database or its structures.
- Automatically committed; made permanent.

Data Manipulation Language: DML commands are a set of commands used to manage data in database objects. It allows the user to insert, update, delete, and retrieve data.
Features

- Modifies the data in the tables.
- Changes are rollbacks and are not permanent unless they have been committed.

Transaction Control Language: TCL commands are those that control transactions occurring within a database. These handles ensuring the integrity of data while committing or rolling back from changes.
Features

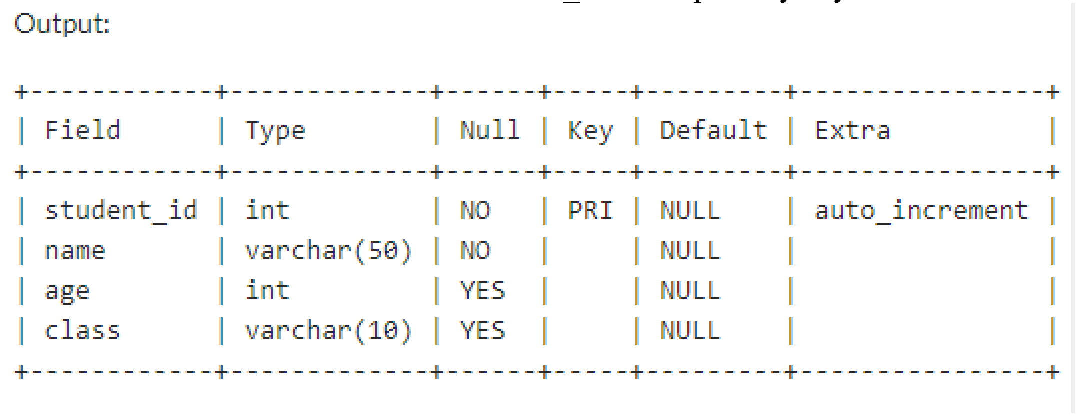
- Works based on transactions.
- Used to maintain consistency in databases.

Queries: To be perform in MySQL Database.

DDL (Data Definition Language) Queries

1. **Create a Table**

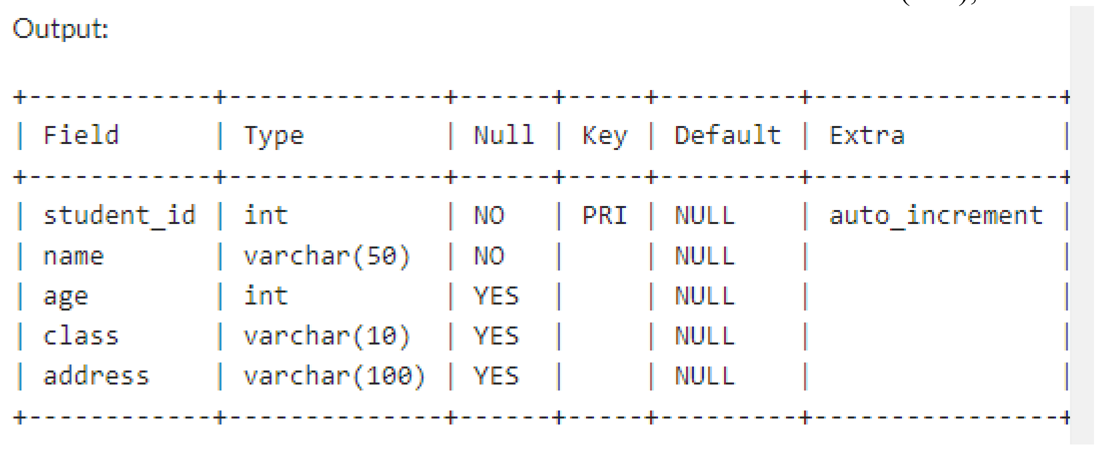
   CREATE TABLE students (student_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50) NOT NULL, age INT, class VARCHAR(10));
   Note: Creates a student's table with student_id as the primary key.



```
Output:

+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| student_id | int         | NO   | PRI | NULL    | auto_increment |
| name       | varchar(50) | NO   |     | NULL    |                |
| age        | int         | YES  |     | NULL    |                |
| class      | varchar(10) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

2. Alter a Table

ALTER TABLE students ADD COLUMN address VARCHAR(100);
Note: ALTER TABLE students ADD COLUMN address VARCHAR(100);

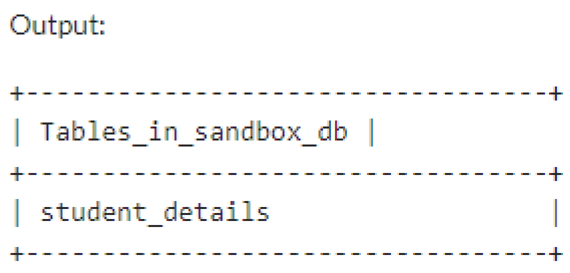

Output:

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| student_id | int          | NO   | PRI | NULL    | auto_increment |
| name       | varchar(50)  | NO   |     | NULL    |                |
| age        | int          | YES  |     | NULL    |                |
| class      | varchar(10)  | YES  |     | NULL    |                |
| address    | varchar(100) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

3. Rename a Table

RENAME TABLE students TO student_details;

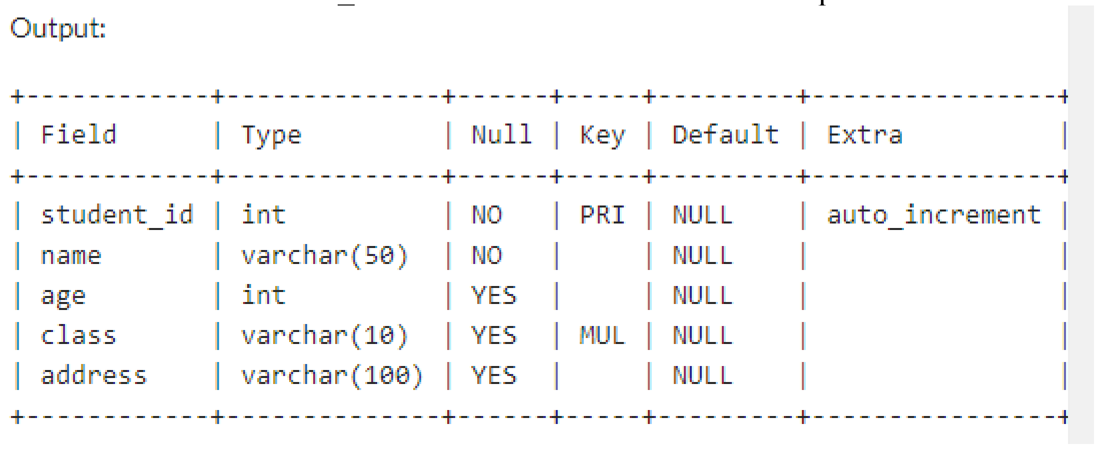
Note: Renames the table students to student_details.


Output:

```
+--------------------------------+
| Tables_in_sandbox_db |
+--------------------------------+
| student_details                |
+--------------------------------+
```

4. Create an Index

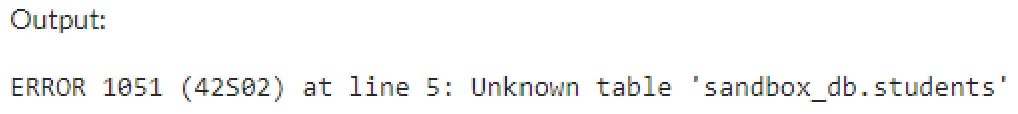
CREATE INDEX idx_class ON student_details(class);
Note: Creates an index idx_class on the class column for faster queries.


Output:

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| student_id | int          | NO   | PRI | NULL    | auto_increment |
| name       | varchar(50)  | NO   |     | NULL    |                |
| age        | int          | YES  |     | NULL    |                |
| class      | varchar(10)  | YES  | MUL | NULL    |                |
| address    | varchar(100) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

5. **Drop a Table**

   DROP TABLE students;
   Note: Deletes the students table if it exists.



Output:

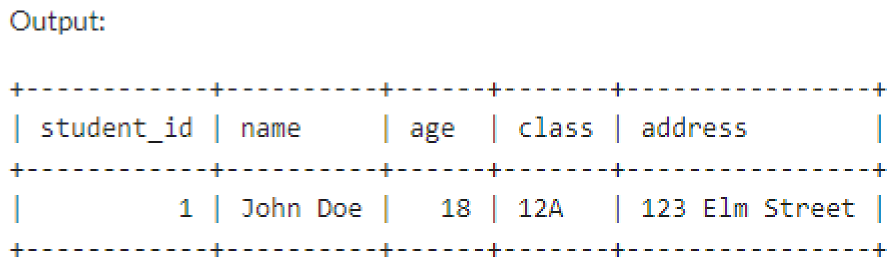
```
ERROR 1051 (42S02) at line 5: Unknown table 'sandbox_db.students'
```

   As the students table was renamed, it is showing the above error.

DML (Data Manipulation Language) Queries

1. **Insert Data**

   INSERT INTO student_details (name, age, class, address) VALUES ('John Doe', 18, '12A', '123 Elm Street');

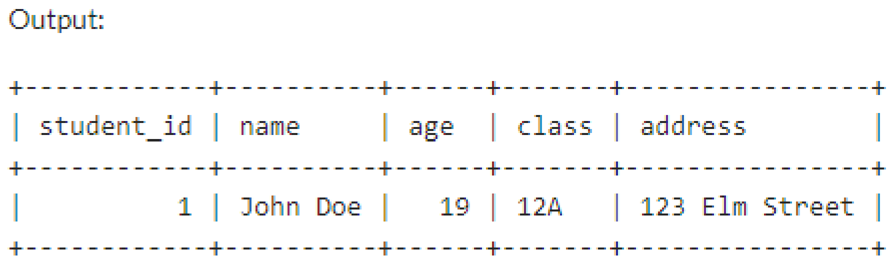


Output:

```
+------------+----------+------+-------+----------------+
| student_id | name     | age  | class | address        |
+------------+----------+------+-------+----------------+
|          1 | John Doe |   18 | 12A   | 123 Elm Street |
+------------+----------+------+-------+----------------+
```

2. **Update Data**

   UPDATE student_details SET age = 19 WHERE name = 'John Doe';

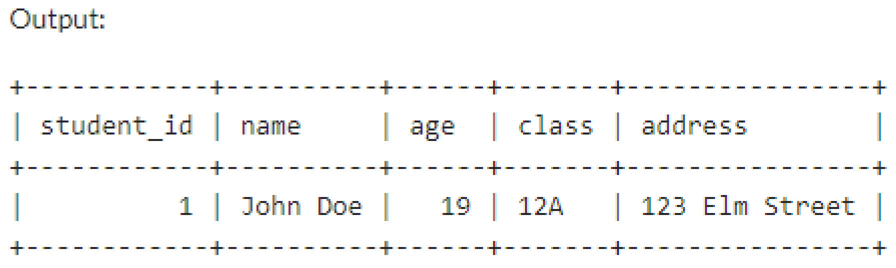


Output:

```
+------------+----------+------+-------+----------------+
| student_id | name     | age  | class | address        |
+------------+----------+------+-------+----------------+
|          1 | John Doe |   19 | 12A   | 123 Elm Street |
+------------+----------+------+-------+----------------+
```

3. **Delete Data**

   DELETE FROM student_details WHERE age < 18;



Output:

```
+------------+----------+------+-------+----------------+
| student_id | name     | age  | class | address        |
+------------+----------+------+-------+----------------+
|          1 | John Doe |   19 | 12A   | 123 Elm Street |
+------------+----------+------+-------+----------------+
```

4. **Select Data**

SELECT * FROM student_details WHERE class = '12A';

Output:

```
+------------+----------+------+-------+----------------+
| student_id | name     | age  | class | address        |
+------------+----------+------+-------+----------------+
|          1 | John Doe |   19 | 12A   | 123 Elm Street |
+------------+----------+------+-------+----------------+
```

5. **Insert Multiple Rows**

   INSERT INTO student_details (name, age, class, address) VALUES ('Jane Smith', 17, '11B', '456 Oak Avenue'), ('Michael Brown', 18, '12A', '789 Pine Road');

   Output:

```
+------------+---------------+------+-------+----------------+
| student_id | name          | age  | class | address        |
+------------+---------------+------+-------+----------------+
|          1 | John Doe      |   19 | 12A   | 123 Elm Street |
|          2 | Jane Smith    |   17 | 11B   | 456 Oak Avenue |
|          3 | Michael Brown |   18 | 12A   | 789 Pine Road  |
+------------+---------------+------+-------+----------------+
```

TCL (Transaction Control Language) Queries

1. **Start a Transaction**

   START TRANSACTION;

2. **Insert and Rollback**

   INSERT INTO student_details (name, age, class, address) VALUES ('Temporary Student', 20, '12C', '999 Test Road');
   ROLLBACK;

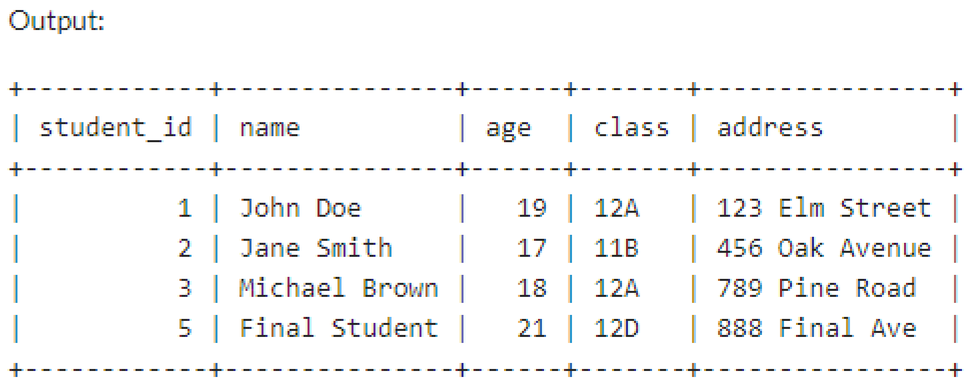
   Output:

```
+------------+---------------+------+-------+----------------+
| student_id | name          | age  | class | address        |
+------------+---------------+------+-------+----------------+
|          1 | John Doe      |   19 | 12A   | 123 Elm Street |
|          2 | Jane Smith    |   17 | 11B   | 456 Oak Avenue |
|          3 | Michael Brown |   18 | 12A   | 789 Pine Road  |
+------------+---------------+------+-------+----------------+
```

   As we used the ROLLBACK command nothing got changed in the table

3. **Insert and Commit**

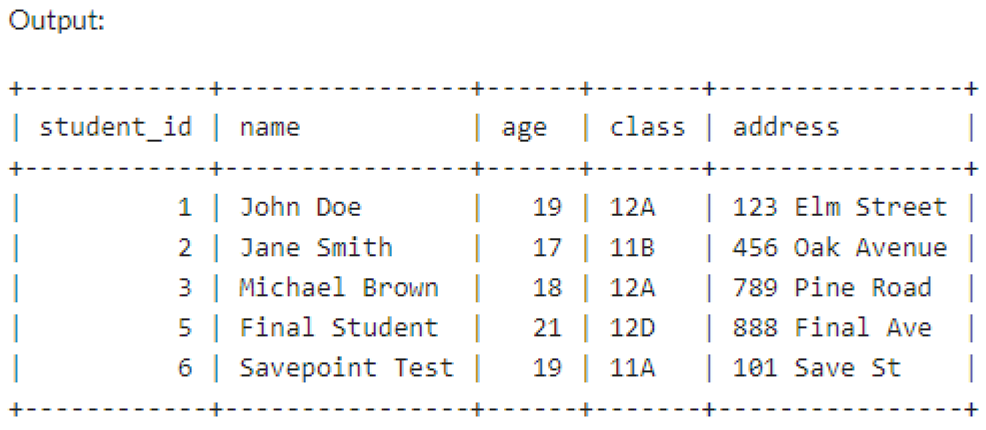
START TRANSACTION;
INSERT INTO student_details (name, age, class, address) VALUES ('Final Student', 21, '12D', '888 Final Ave');
COMMIT;


Output:

```
+------------+--------------+------+-------+---------------+
| student_id | name         | age  | class | address       |
+------------+--------------+------+-------+---------------+
|          1 | John Doe     |   19 | 12A   | 123 Elm Street |
|          2 | Jane Smith   |   17 | 11B   | 456 Oak Avenue |
|          3 | Michael Brown |   18 | 12A   | 789 Pine Road  |
|          5 | Final Student |   21 | 12D   | 888 Final Ave  |
+------------+--------------+------+-------+---------------+
```

4. Savepoint and Rollback

START TRANSACTION;
INSERT INTO student_details (name, age, class, address) VALUES ('Savepoint Test', 19, '11A', '101 Save St');
SAVEPOINT sp1;
INSERT INTO student_details (name, age, class, address) VALUES ('Undo This', 20, '11B', '202 Undo Ave');
ROLLBACK TO sp1;
COMMIT;


Output:

```
+------------+----------------+------+-------+---------------+
| student_id | name           | age  | class | address       |
+------------+----------------+------+-------+---------------+
|          1 | John Doe       |   19 | 12A   | 123 Elm Street |
|          2 | Jane Smith     |   17 | 11B   | 456 Oak Avenue |
|          3 | Michael Brown  |   18 | 12A   | 789 Pine Road  |
|          5 | Final Student  |   21 | 12D   | 888 Final Ave  |
|          6 | Savepoint Test |   19 | 11A   | 101 Save St    |
+------------+----------------+------+-------+---------------+
```

5. Set Transaction Isolation Level

START TRANSACTION;
SELECT * FROM student_details WHERE class = '12A';
COMMIT;

Output:

```
+------------+---------------+------+-------+-----------------+
| student_id | name          | age  | class | address         |
+------------+---------------+------+-------+-----------------+
|          1 | John Doe      |   19 | 12A   | 123 Elm Street  |
|          3 | Michael Brown |   18 | 12A   | 789 Pine Road   |
+------------+---------------+------+-------+-----------------+
```

Practical No. 2

Aim: Hadoop Shell Commands

Explanation:

Introduction

Apache Hadoop is an open-source framework that assists in the processing, storage, and analysis of voluminous digital information across distributed systems. Created by the Apache Software Foundation, Hadoop is scalable from one to thousands of machines where fault-tolerant data processing can be done.

Core Components

HDFS (Hadoop Distributed File System):

This is a distributed storage architecture for storing large files spanning across a number of machines. Other features include high fault tolerance, scalability, and its optimization for read-intensive processing.

MapReduce:

An application programming model for the parallel process of data.

Steps:

- Map Phase: input data processing, which results in producing key-value pairs.
- Reduce Phase: aggregation and further processing of output obtained from the map phase.

YARN (Yet Another Resource Negotiator):

The resource management system uses data from the underlying file system. Provides resource scheduling and subsequent allocation of these resources to multi-user applications.

Hadoop Common:

A set of utilities and libraries that other Hadoop modules require.

Features

- **Scalability:** processes petabytes of information.
- **Cost-Effective:** accommodates the consumer commodity.
- **Fault Tolerance:** replication of data.
- **Versatility:** varied types of data format, both structured, semi-structured, and unstructured.

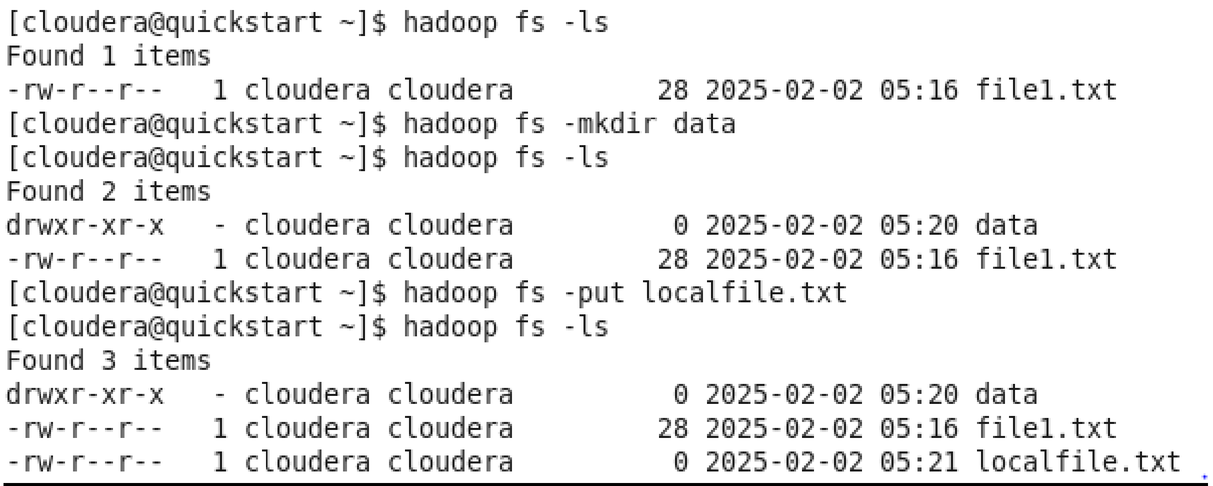
Shell Commands: Using Hadoop in Cloudera file the following shell commands

1. **hadoop fs –ls**

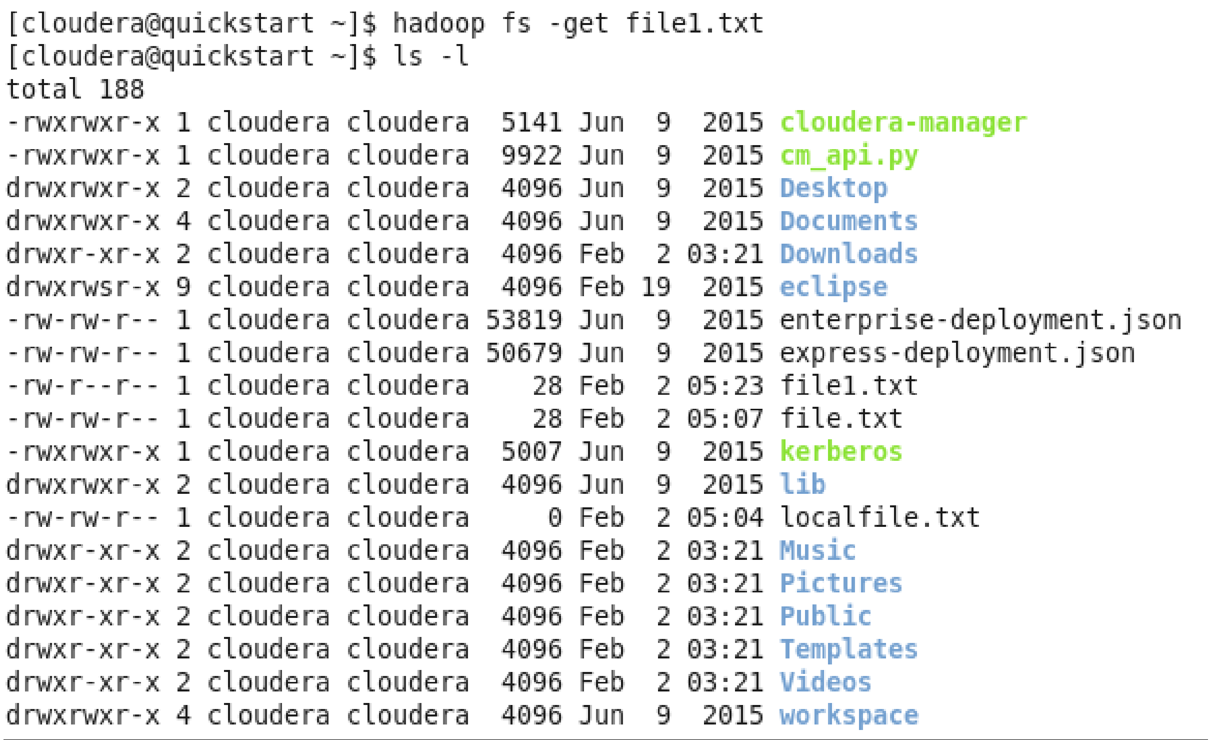
- Lists the contents of a directory in HDFS.
- Example: hadoop fs -ls /user/hadoop/

2. **hadoop fs –mkdir**
   - Creates directories in HDFS.
   - Example: hadoop fs -mkdir data
3. **hadoop fs –put**
   - Uploads a file or directory from the local filesystem to HDFS.
   - Example: hadoop fs -put localfile.txt
4. **hadoop fs –get**
   - Downloads files or directories from HDFS to the local filesystem.
   - Example: hadoop fs -get data.txt
5. **hadoop fs –rm**
   - Removes files or directories from HDFS.
   - Example: hadoop fs -rm data.txt
6. **hadoop fs –rmdir**
   - Removes an empty directory from HDFS.
   - Example: hadoop fs -rmdir emptydir
7. **hadoop fs –cat**
   - Displays the contents of a file in HDFS.
   - Example: hadoop fs -cat file.txt
8. **hadoop fs –mv**
   - Moves files or directories within HDFS.
   - Example: hadoop fs -mv file1.txt renamed_file1.txt
9. **hadoop fs –cp**
   - Copies files or directories within HDFS.
   - Example: hadoop fs -cp file1.txt file2.txt
10. **hadoop fs –du**
    - Shows the disk usage of files and directories in HDFS.
    - Example: hadoop fs -du
11. **hadoop fs –count**
    - Displays file and directory count, space used, and space available in HDFS.
    - Example: hadoop fs -count
12. **hadoop fs –tail**
    - Shows the last part of a file in HDFS.
    - Example: hadoop fs -tail logs.txt

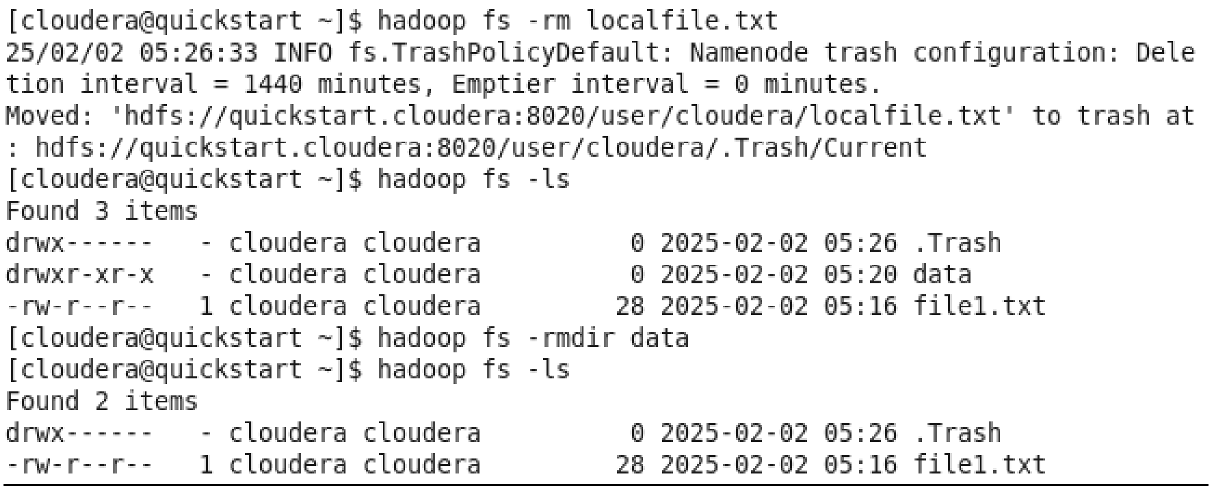
Output:



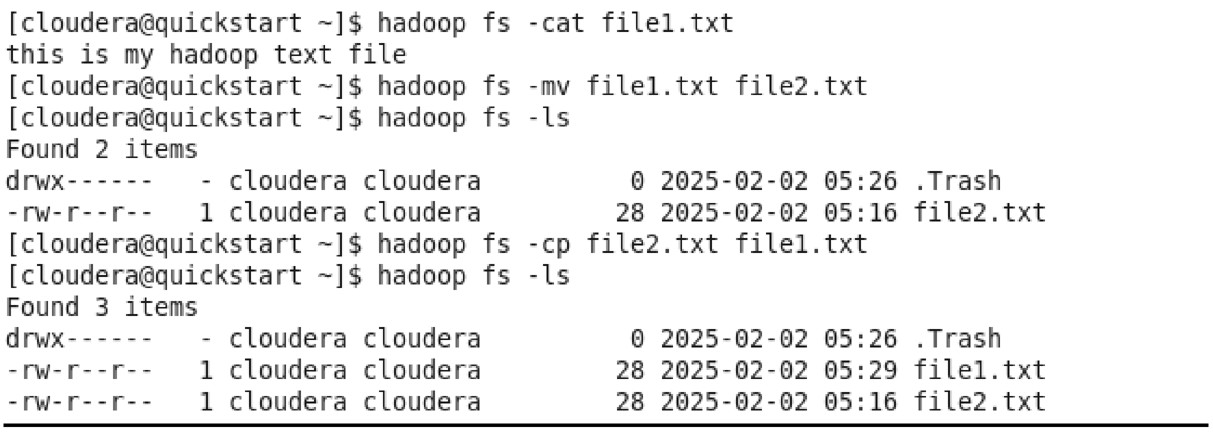
```
[cloudera@quickstart ~]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file1.txt
[cloudera@quickstart ~]$ hadoop fs -mkdir data
[cloudera@quickstart ~]$ hadoop fs -ls
Found 2 items
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 05:20 data
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file1.txt
[cloudera@quickstart ~]$ hadoop fs -put localfile.txt
[cloudera@quickstart ~]$ hadoop fs -ls
Found 3 items
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 05:20 data
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file1.txt
-rw-r--r--   1 cloudera cloudera          0 2025-02-02 05:21 localfile.txt
```



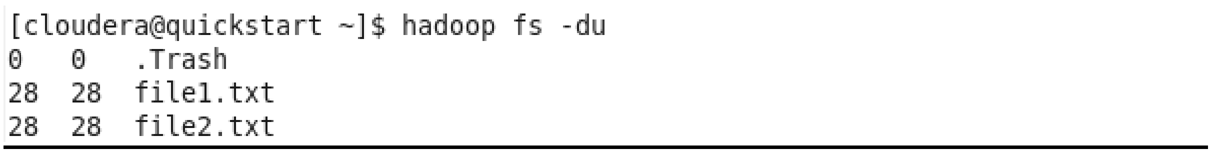
```
[cloudera@quickstart ~]$ hadoop fs -get file1.txt
[cloudera@quickstart ~]$ ls -l
total 188
-rwxrwxr-x 1 cloudera cloudera  5141 Jun  9  2015 cloudera-manager
-rwxrwxr-x 1 cloudera cloudera  9922 Jun  9  2015 cm_api.py
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 Desktop
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 Documents
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Downloads
drwxrwsr-x 9 cloudera cloudera  4096 Feb 19  2015 eclipse
-rw-rw-r-- 1 cloudera cloudera 53819 Jun  9  2015 enterprise-deployment.json
-rw-rw-r-- 1 cloudera cloudera 50679 Jun  9  2015 express-deployment.json
-rw-r--r-- 1 cloudera cloudera    28 Feb  2 05:23 file1.txt
-rw-rw-r-- 1 cloudera cloudera    28 Feb  2 05:07 file.txt
-rwxrwxr-x 1 cloudera cloudera  5007 Jun  9  2015 kerberos
drwxrwxr-x 2 cloudera cloudera  4096 Jun  9  2015 lib
-rw-rw-r-- 1 cloudera cloudera     0 Feb  2 05:04 localfile.txt
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Music
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Pictures
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Public
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Templates
drwxr-xr-x 2 cloudera cloudera  4096 Feb  2 03:21 Videos
drwxrwxr-x 4 cloudera cloudera  4096 Jun  9  2015 workspace
```



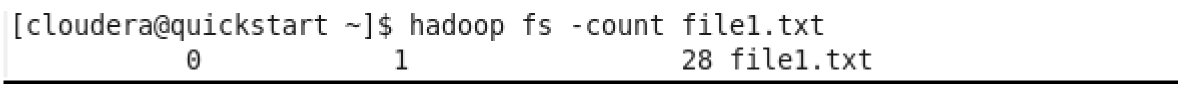
```
[cloudera@quickstart ~]$ hadoop fs -rm localfile.txt
25/02/02 05:26:33 INFO fs.TrashPolicyDefault: Namenode trash configuration: Dele
tion interval = 1440 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://quickstart.cloudera:8020/user/cloudera/localfile.txt' to trash at
: hdfs://quickstart.cloudera:8020/user/cloudera/.Trash/Current
[cloudera@quickstart ~]$ hadoop fs -ls
Found 3 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 05:20 data
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file1.txt
[cloudera@quickstart ~]$ hadoop fs -rmdir data
[cloudera@quickstart ~]$ hadoop fs -ls
Found 2 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file1.txt
```



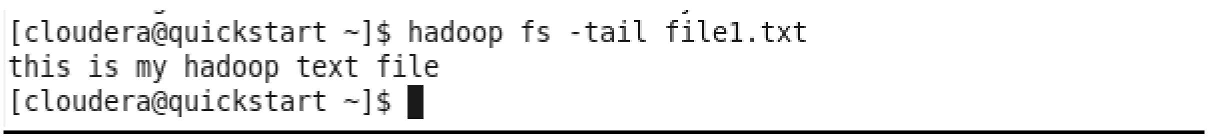
```
[cloudera@quickstart ~]$ hadoop fs -cat file1.txt
this is my hadoop text file
[cloudera@quickstart ~]$ hadoop fs -mv file1.txt file2.txt
[cloudera@quickstart ~]$ hadoop fs -ls
Found 2 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file2.txt
[cloudera@quickstart ~]$ hadoop fs -cp file2.txt file1.txt
[cloudera@quickstart ~]$ hadoop fs -ls
Found 3 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:29 file1.txt
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file2.txt
```



```
[cloudera@quickstart ~]$ hadoop fs -du
0   0   .Trash
28  28  file1.txt
28  28  file2.txt
```



```
[cloudera@quickstart ~]$ hadoop fs -count file1.txt
           0            1                 28 file1.txt
```



```
[cloudera@quickstart ~]$ hadoop fs -tail file1.txt
this is my hadoop text file
[cloudera@quickstart ~]$ 
```

Practical No. 3

Aim: Wordcount implementation using Pig

Explanation:
Apache Pig is a high-level platform to analyze and process large datasets using Hadoop. It abstracts the complexity of writing MapReduce programs and provides a simple scripting language known as Pig Latin. This Pig Latin script conveniently lets the user perform complex data-realignment and data transformations without writing any Java code.

Some Major Features of Pig:

- **Easy to write:** It is easier to express a complex logical data transformation in Pig Latin than in Java.
- **Extensible:** It supports the writing of user-defined functions (UDFs) in Java, Python, and Ruby.
- **Optimization:** Pig optimizes the execution by converting Pig Latin script to MapReduce jobs.
- **Interoperability:** It can deal with both structured and unstructured data, thus is applicable to a number of types of analysis.

Pig Components:

- **Pig Latin Script:** The high-level language for describing the data processing task.
- **Pig Compiler:** Translates Pig Latin scripts into a series of MapReduce jobs for execution on Hadoop.

Execution modes:

- **Local Mode:** obviously invoked directly on the single machine, under very controlled conditions.
- **Hadoop Mode:** for executing scripts on a Hadoop cluster via the HDFS.

Pros:

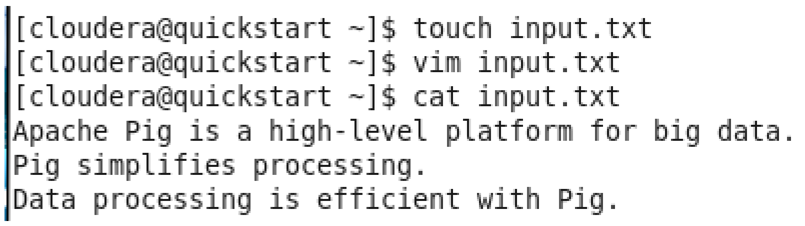
- Development time saved is a major benefit.
- More scalable-relatively easy to handle a large dataset than native MapReduce programming.
- Flexibility-relatively easy to support different data formats like JSON, XML, and plain text.

Wordcount implementation

Using Pig in Cloudera implement the following Code for Word count:

1. **Create a Text File:**

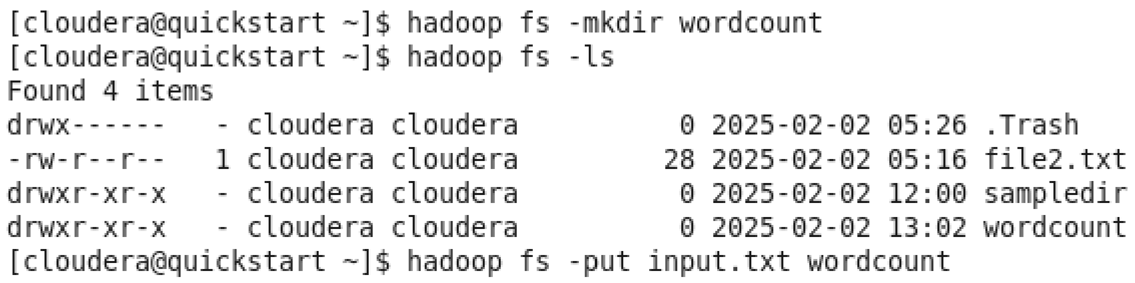
   Create a file input.txt with some sample text.
   E.g :-
   Apache Pig is a high-level platform for big data.
   Pig simplifies processing.
   Data processing is efficient with Pig.



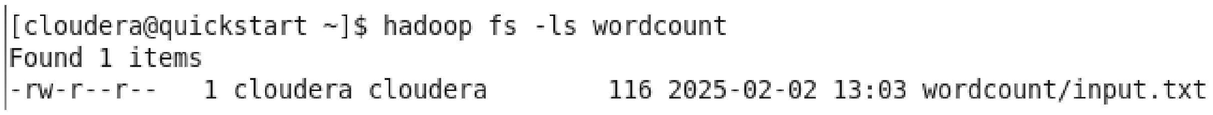
```
[cloudera@quickstart ~]$ touch input.txt
[cloudera@quickstart ~]$ vim input.txt
[cloudera@quickstart ~]$ cat input.txt
Apache Pig is a high-level platform for big data.
Pig simplifies processing.
Data processing is efficient with Pig.
```

2. **Copy the File to HDFS: Open a terminal and execute**

   hadoop fs -mkdir /user/cloudera/wordcount
   hadoop fs -put input.txt /user/cloudera/wordcount



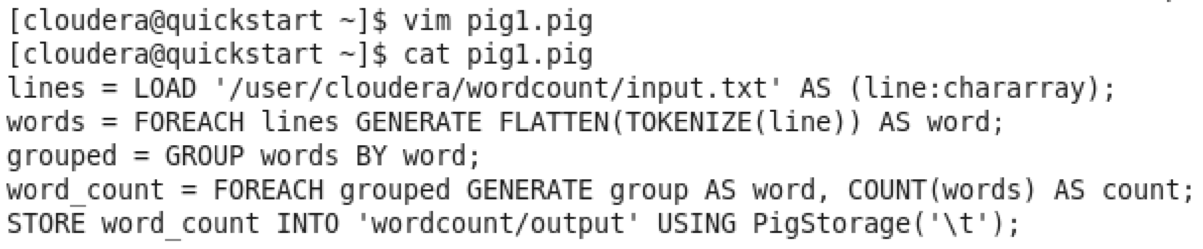
```
[cloudera@quickstart ~]$ hadoop fs -mkdir wordcount
[cloudera@quickstart ~]$ hadoop fs -ls
Found 4 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file2.txt
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 12:00 sampledir
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 13:02 wordcount
[cloudera@quickstart ~]$ hadoop fs -put input.txt wordcount
```



```
[cloudera@quickstart ~]$ hadoop fs -ls wordcount
Found 1 items
-rw-r--r--   1 cloudera cloudera        116 2025-02-02 13:03 wordcount/input.txt
```

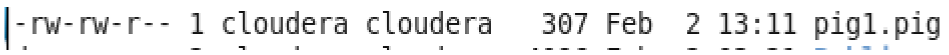
3. **Type the code using text editor**

   lines = LOAD '/user/cloudera/wordcount/input.txt' AS (line:chararray);
   words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
   grouped = GROUP words BY word;
   word_count = FOREACH grouped GENERATE group AS word, COUNT(words) AS count;
   STORE word_count INTO 'wordcount/output' USING PigStorage('\t');



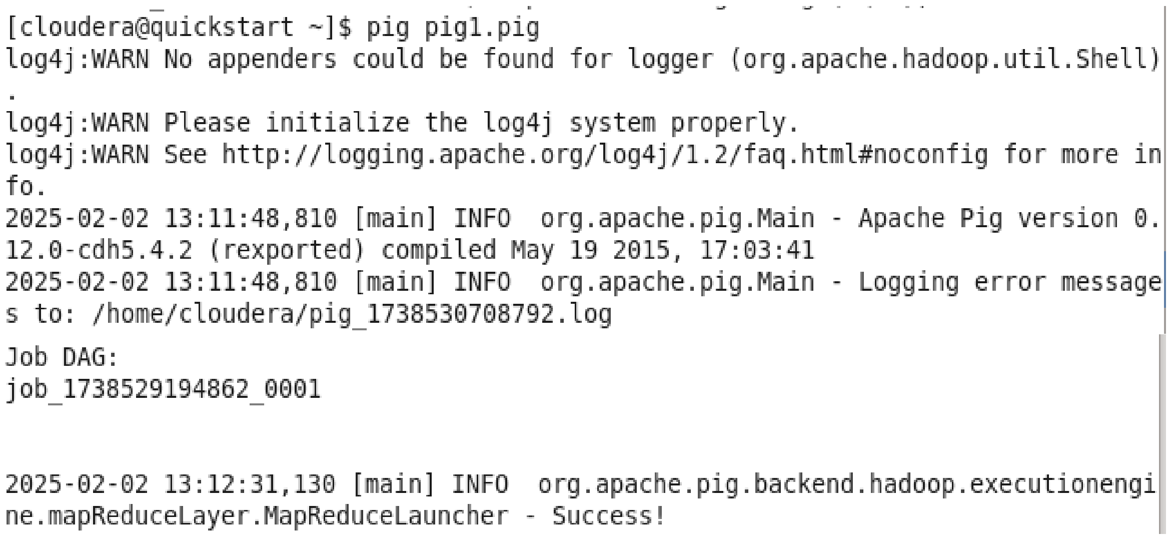
```
[cloudera@quickstart ~]$ vim pig1.pig
[cloudera@quickstart ~]$ cat pig1.pig
lines = LOAD '/user/cloudera/wordcount/input.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
grouped = GROUP words BY word;
word_count = FOREACH grouped GENERATE group AS word, COUNT(words) AS count;
STORE word_count INTO 'wordcount/output' USING PigStorage('\t');
```

4. **Save the file as pig1.pig**



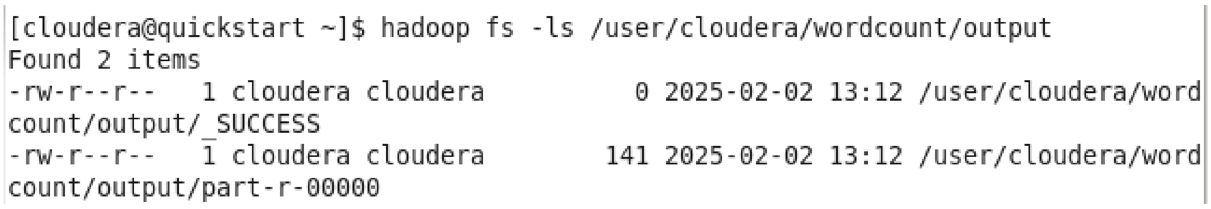
```
-rw-rw-r-- 1 cloudera cloudera   307 Feb  2 13:11 pig1.pig
```

5. **Execute the pig script:**
   a. Make sure Hadoop and yarn are active in cloudera
   b. Using the terminal execute the command:
      pig pig1.pig



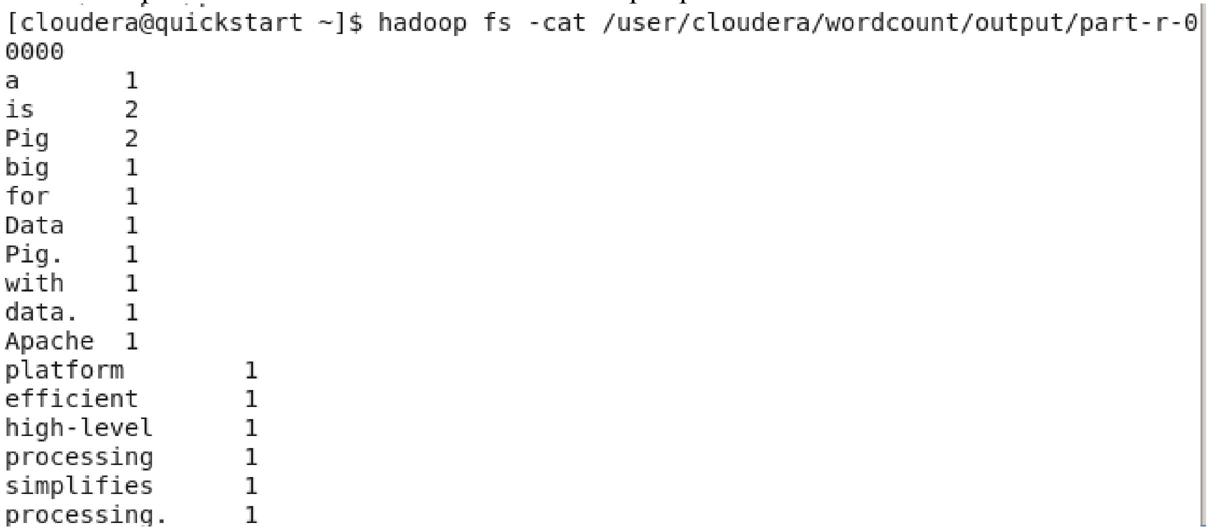
```
[cloudera@quickstart ~]$ pig pig1.pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell)
.
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
2025-02-02 13:11:48,810 [main] INFO  org.apache.pig.Main - Apache Pig version 0.
12.0-cdh5.4.2 (rexported) compiled May 19 2015, 17:03:41
2025-02-02 13:11:48,810 [main] INFO  org.apache.pig.Main - Logging error message
s to: /home/cloudera/pig_1738530708792.log
Job DAG:
job_1738529194862_0001


2025-02-02 13:12:31,130 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
```

6. **View the output:**
   hadoop fs -ls /user/cloudera/wordcount/output



```
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera/wordcount/output
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2025-02-02 13:12 /user/cloudera/word
count/output/_SUCCESS
-rw-r--r--   1 cloudera cloudera        141 2025-02-02 13:12 /user/cloudera/word
count/output/part-r-00000
```

   hadoop fs -cat /user/cloudera/wordcount/output/part-r-00000



```
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/wordcount/output/part-r-0
0000
a       1
is      2
Pig     2
big     1
for     1
Data    1
Pig.    1
with    1
data.   1
Apache  1
platform        1
efficient       1
high-level      1
processing      1
simplifies      1
processing.     1
```

Practical No. 4

Aim: Data wrangling using Pig

Explanation:
Data wrangling, also known as data munging, involves transforming and cleaning raw data into a format suitable for analysis. It is a crucial step in the data pipeline for tasks like data integration, enrichment, and preprocessing. Apache Pig, a platform for processing large datasets, is widely used for efficient and scalable data wrangling in Big Data ecosystems. Data wrangling using Apache Pig in Cloudera involves the **extraction, transformation, and loading (ETL)** of large datasets. Pig's high-level scripting language, Pig Latin, is designed to simplify the data manipulation process, making it easier to clean and preprocess data for further analysis.

Set Up Pig in Cloudera

- Ensure that Cloudera is installed and properly configured on your system.
- Start the Hadoop cluster using Cloudera Manager or directly via terminal.
- Open the Pig Grunt shell

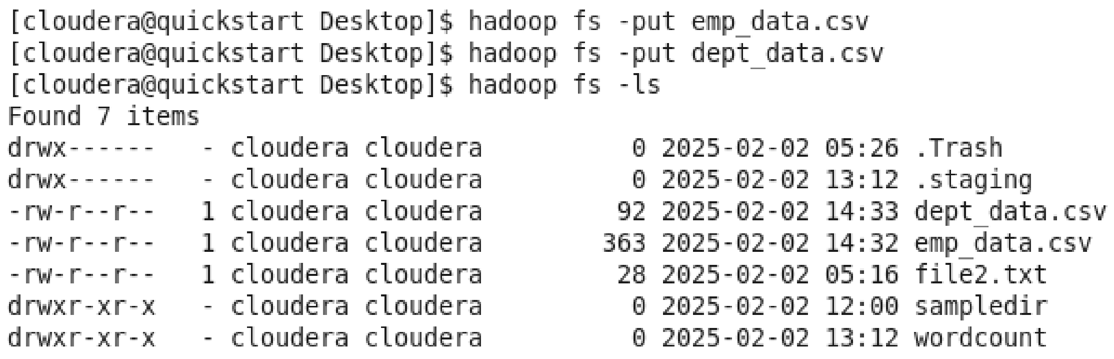
Create a CSV file dataset using Text Editor and save its as emp_data.csv

```
id,name,age,city,salary
1,John Doe,28,New York,-50000
2,Jane Smith,34,Los Angeles,60000
3,Mike Johnson,25,Chicago,45000.45
4,Linda Green,40,New York,75000.55
5,James White,29,Los Angeles,55000
6,Patricia Brown,50,Chicago,-80000
7,Robert Black,,New York,48000
8,Emily Davis,27,Chicago,49000
9,William Harris,35,Los Angeles,62000
10,Elizabeth Clark,30,New York,52000
```

Create a CSV file dataset using Text Editor and save its as dept_data.csv

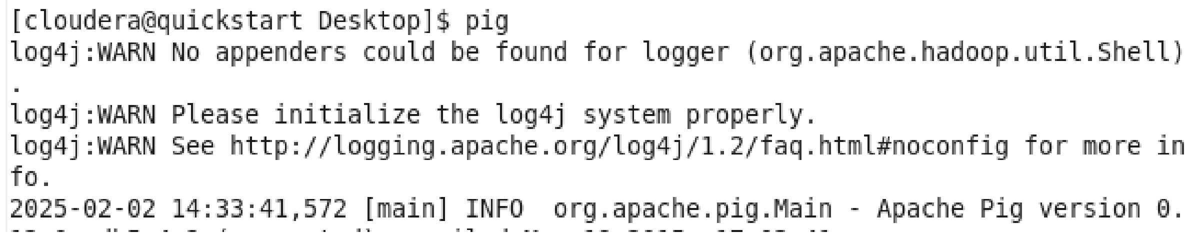
```
id,dept
1,Sales
2,Marketing
3,IT
4,HR
5,Finance
6,Operations
7,Sales
8,IT
9,Marketing
10,HR
```

1. **Upload the CSV files to hadoop file system.**



```
[cloudera@quickstart Desktop]$ hadoop fs -put emp_data.csv
[cloudera@quickstart Desktop]$ hadoop fs -put dept_data.csv
[cloudera@quickstart Desktop]$ hadoop fs -ls
Found 7 items
drwx------   - cloudera cloudera          0 2025-02-02 05:26 .Trash
drwx------   - cloudera cloudera          0 2025-02-02 13:12 .staging
-rw-r--r--   1 cloudera cloudera         92 2025-02-02 14:33 dept_data.csv
-rw-r--r--   1 cloudera cloudera        363 2025-02-02 14:32 emp_data.csv
-rw-r--r--   1 cloudera cloudera         28 2025-02-02 05:16 file2.txt
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 12:00 sampledir
drwxr-xr-x   - cloudera cloudera          0 2025-02-02 13:12 wordcount
```

2. **Start Apache Pig by entering pig command.**



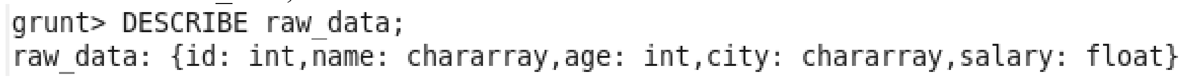
```
[cloudera@quickstart Desktop]$ pig
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell)
.
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more in
fo.
2025-02-02 14:33:41,572 [main] INFO  org.apache.pig.Main - Apache Pig version 0.
```

3. **Use the LOAD function to load data from HDFS or a local file.**

raw_data = LOAD 'emp_data.csv' USING PigStorage(',') AS (id:int, name:chararray, age:int, city:chararray, salary:float);

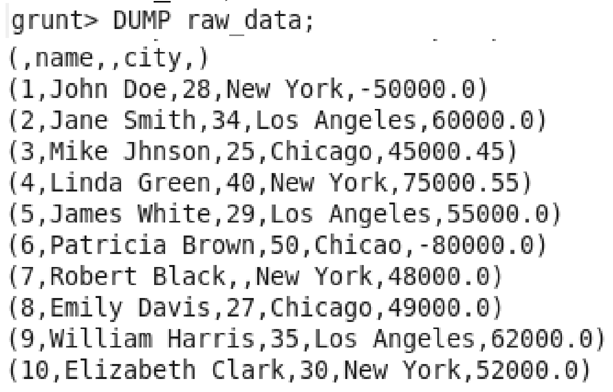
4. **Inspect the Data**

Use the DUMP or DESCRIBE commands to understand the schema and verify the data
DESCRIBE raw_data;



```
grunt> DESCRIBE raw_data;
raw_data: {id: int,name: chararray,age: int,city: chararray,salary: float}
```

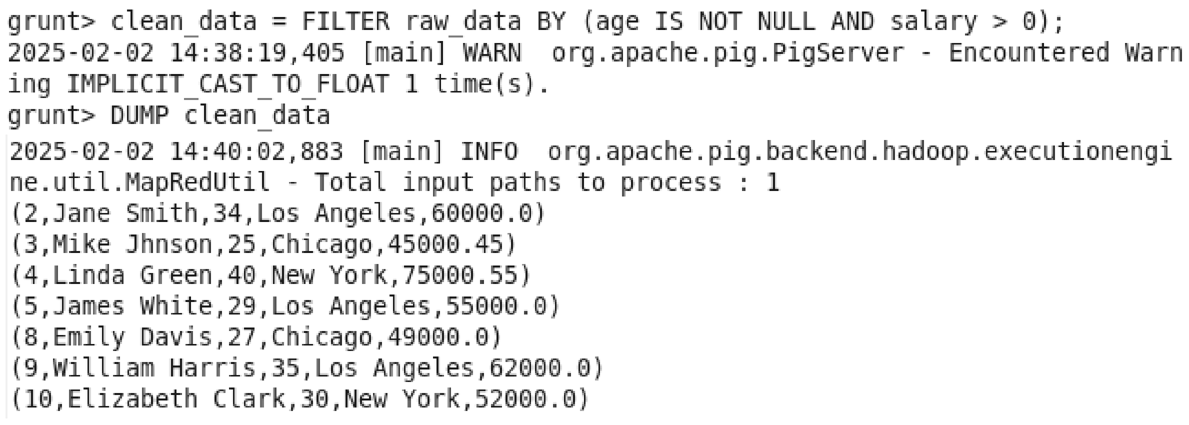
DUMP raw_data;



```
grunt> DUMP raw_data;
(,name,,city,)
(1,John Doe,28,New York,-50000.0)
(2,Jane Smith,34,Los Angeles,60000.0)
(3,Mike Jhnson,25,Chicago,45000.45)
(4,Linda Green,40,New York,75000.55)
(5,James White,29,Los Angeles,55000.0)
(6,Patricia Brown,50,Chicao,-80000.0)
(7,Robert Black,,New York,48000.0)
(8,Emily Davis,27,Chicago,49000.0)
(9,William Harris,35,Los Angeles,62000.0)
(10,Elizabeth Clark,30,New York,52000.0)
```

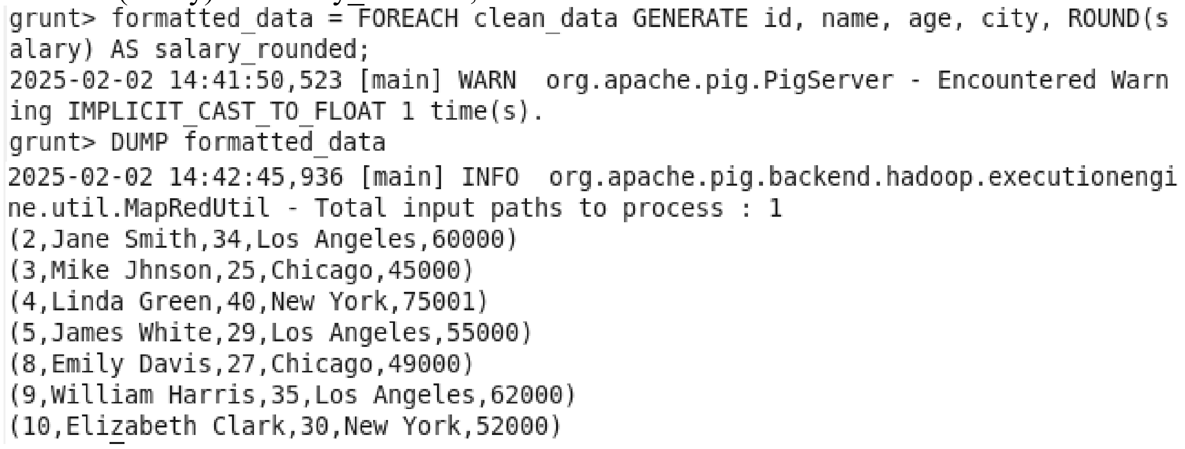
5. **Data Cleaning**

Remove Null or Invalid Entries:
clean_data = FILTER raw_data BY (age IS NOT NULL AND salary > 0);



```
grunt> clean_data = FILTER raw_data BY (age IS NOT NULL AND salary > 0);
2025-02-02 14:38:19,405 [main] WARN  org.apache.pig.PigServer - Encountered Warn
ing IMPLICIT_CAST_TO_FLOAT 1 time(s).
grunt> DUMP clean_data
2025-02-02 14:40:02,883 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2,Jane Smith,34,Los Angeles,60000.0)
(3,Mike Jhnson,25,Chicago,45000.45)
(4,Linda Green,40,New York,75000.55)
(5,James White,29,Los Angeles,55000.0)
(8,Emily Davis,27,Chicago,49000.0)
(9,William Harris,35,Los Angeles,62000.0)
(10,Elizabeth Clark,30,New York,52000.0)
```

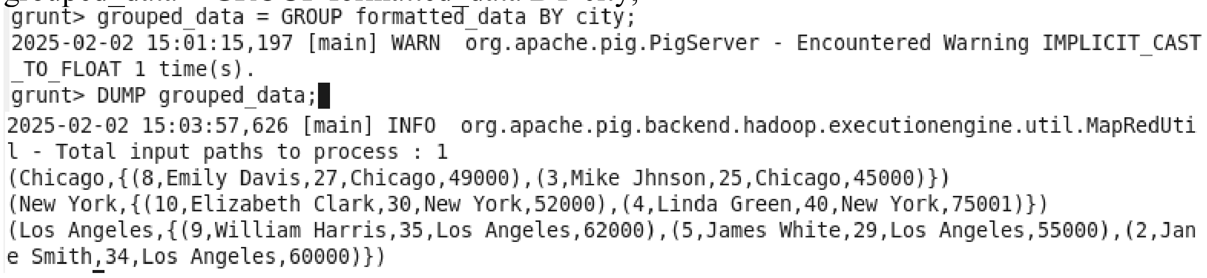
Format Data: Use functions like UPPER, LOWER, TRIM, etc
formatted_data = FOREACH clean_data GENERATE id, name, age, city, ROUND(salary) AS salary_rounded;



```
grunt> formatted_data = FOREACH clean_data GENERATE id, name, age, city, ROUND(s
alary) AS salary_rounded;
2025-02-02 14:41:50,523 [main] WARN  org.apache.pig.PigServer - Encountered Warn
ing IMPLICIT_CAST_TO_FLOAT 1 time(s).
grunt> DUMP formatted_data
2025-02-02 14:42:45,936 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2,Jane Smith,34,Los Angeles,60000)
(3,Mike Jhnson,25,Chicago,45000)
(4,Linda Green,40,New York,75001)
(5,James White,29,Los Angeles,55000)
(8,Emily Davis,27,Chicago,49000)
(9,William Harris,35,Los Angeles,62000)
(10,Elizabeth Clark,30,New York,52000)
```

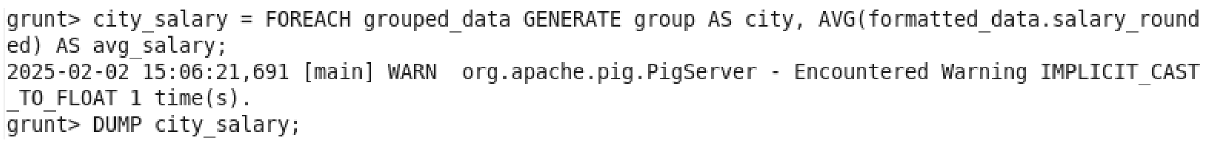
6. **Data Transformation**

Group Data:
grouped_data = GROUP formatted_data BY city;

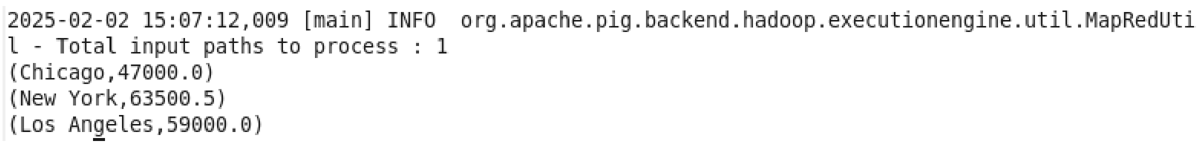


```
grunt> grouped_data = GROUP formatted_data BY city;
2025-02-02 15:01:15,197 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST
_TO_FLOAT 1 time(s).
grunt> DUMP grouped_data;
2025-02-02 15:03:57,626 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l - Total input paths to process : 1
(Chicago,{(8,Emily Davis,27,Chicago,49000),(3,Mike Jhnson,25,Chicago,45000)})
(New York,{(10,Elizabeth Clark,30,New York,52000),(4,Linda Green,40,New York,75001)})
(Los Angeles,{(9,William Harris,35,Los Angeles,62000),(5,James White,29,Los Angeles,55000),(2,Jan
e Smith,34,Los Angeles,60000)})
```

Aggregate Data:
city_salary = FOREACH grouped_data GENERATE group AS city, AVG(formatted_data.salary_rounded) AS avg_salary;



```
grunt> city_salary = FOREACH grouped_data GENERATE group AS city, AVG(formatted_data.salary_round
ed) AS avg_salary;
2025-02-02 15:06:21,691 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST
_TO_FLOAT 1 time(s).
grunt> DUMP city_salary;
```



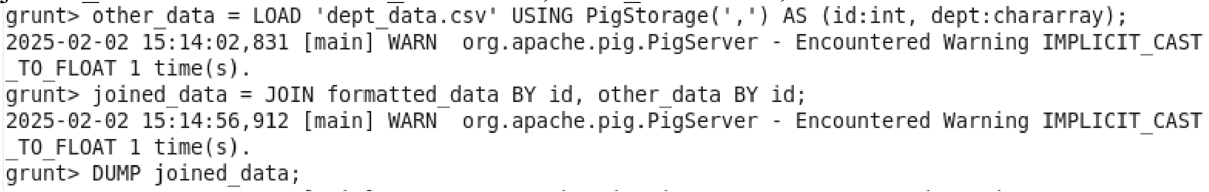
```
2025-02-02 15:07:12,009 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l - Total input paths to process : 1
(Chicago,47000.0)
(New York,63500.5)
(Los Angeles,59000.0)
```

7. **Store Results**

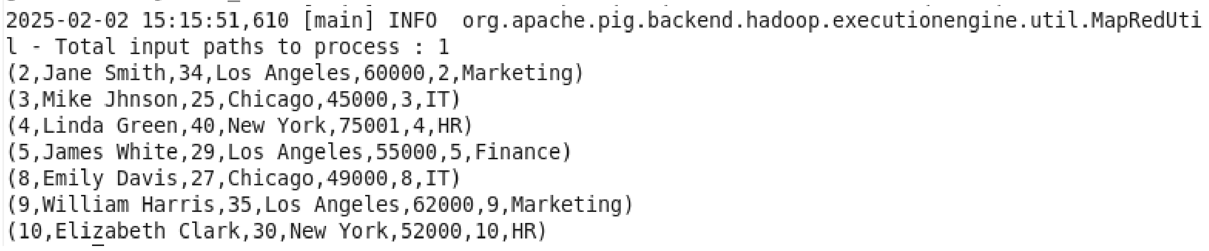
   Save the processed data back to HDFS or export it to a file.
   STORE city_salary INTO 'emp_data' USING PigStorage(',');

8. **Join Datasets:**

   other_data = LOAD ‘dept_data.csv' USING PigStorage(',') AS (id:int, dept:chararray);
   joined_data = JOIN formatted_data BY id, other_data BY id;



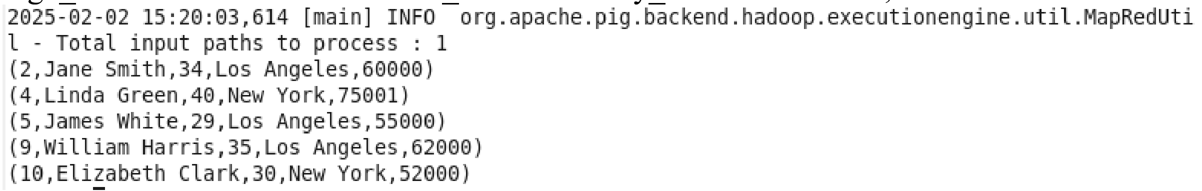
```
grunt> other_data = LOAD 'dept_data.csv' USING PigStorage(',') AS (id:int, dept:chararray);
2025-02-02 15:14:02,831 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST
_TO_FLOAT 1 time(s).
grunt> joined_data = JOIN formatted_data BY id, other_data BY id;
2025-02-02 15:14:56,912 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST
_TO_FLOAT 1 time(s).
grunt> DUMP joined_data;
```



```
2025-02-02 15:15:51,610 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l - Total input paths to process : 1
(2,Jane Smith,34,Los Angeles,60000,2,Marketing)
(3,Mike Jhnson,25,Chicago,45000,3,IT)
(4,Linda Green,40,New York,75001,4,HR)
(5,James White,29,Los Angeles,55000,5,Finance)
(8,Emily Davis,27,Chicago,49000,8,IT)
(9,William Harris,35,Los Angeles,62000,9,Marketing)
(10,Elizabeth Clark,30,New York,52000,10,HR)
```

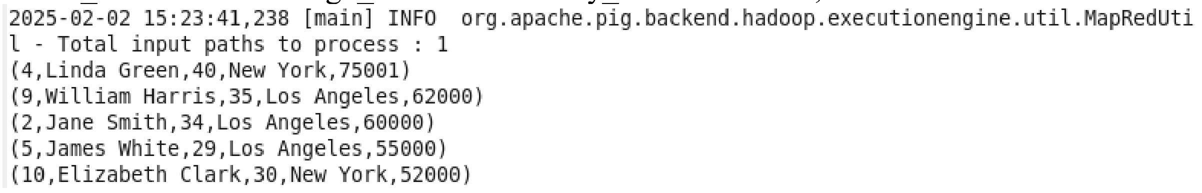
9. **Data Filtering and Sorting**

   Filter Specific Records:
   high_earners = FILTER formatted_data BY salary_rounded > 50000;



```
2025-02-02 15:20:03,614 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l - Total input paths to process : 1
(2,Jane Smith,34,Los Angeles,60000)
(4,Linda Green,40,New York,75001)
(5,James White,29,Los Angeles,55000)
(9,William Harris,35,Los Angeles,62000)
(10,Elizabeth Clark,30,New York,52000)
```

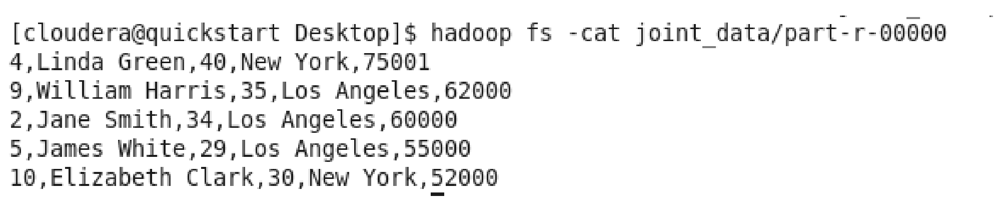
   Sort Data:
   sorted_data = ORDER high_earners BY salary_rounded DESC;



```
2025-02-02 15:23:41,238 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUti
l - Total input paths to process : 1
(4,Linda Green,40,New York,75001)
(9,William Harris,35,Los Angeles,62000)
(2,Jane Smith,34,Los Angeles,60000)
(5,James White,29,Los Angeles,55000)
(10,Elizabeth Clark,30,New York,52000)
```

10. **Store Results**

    Save the processed data back to HDFS or export it to a file.
    STORE sorted_data INTO 'joint_data' USING PigStorage(',');



```
[cloudera@quickstart Desktop]$ hadoop fs -cat joint_data/part-r-00000
4,Linda Green,40,New York,75001
9,William Harris,35,Los Angeles,62000
2,Jane Smith,34,Los Angeles,60000
5,James White,29,Los Angeles,55000
10,Elizabeth Clark,30,New York,52000
```

Practical No. 5

Aim: Create managed and external table in Hive, View in Hadoop Web UI

Explanation:
Apache Hive is a data warehousing tool built on top of Hadoop, designed to facilitate querying, managing, and analyzing large datasets stored in a distributed file system (HDFS). Hive abstracts the complexity of writing MapReduce jobs by providing a SQL-like query language called HiveQL.

Key Features of Hive

1. **SQL-like Language (HiveQL):**
    - Allows users to write queries similar to SQL, which are then translated into MapReduce jobs, Apache Tez, or Apache Spark.

2. **Schema on Read:**
    - Hive applies the schema to data only when a query is executed, rather than when the data is loaded. This makes it ideal for working with unstructured or semi-structured data.

3. **Support for Large Datasets:**
    - Hive can process petabytes of data stored in HDFS or other distributed storage systems.

4. **Integration with Hadoop:**
    - Hive is tightly integrated with Hadoop for distributed storage (HDFS) and distributed processing (MapReduce, Tez, or Spark).

5. **Extensibility:**
    - Supports custom user-defined functions (UDFs) to extend its capabilities.

6. **Support for Various File Formats:**
    - Hive supports a variety of file formats, including: TextFile
        - SequenceFile
        - ORC (Optimized Row Columnar)
        - Parquet
        - AVRO

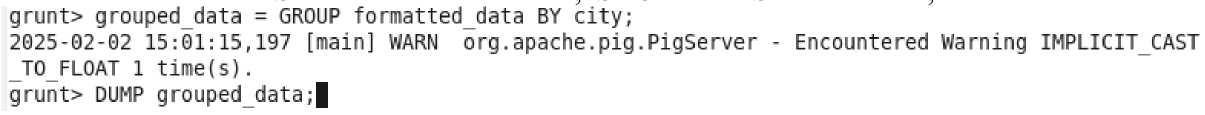
<u>Create Managed and External Table</u>

1. Make sure Hadoop, YARN,ZooKeeper and Hive services are working in Cloudera.
2. Using Command prompt open the hive terminal using the following Command: hive.
3. Using any text editor create the csv file with following data:
   mydata.csv

   1,John Doe,28
   2,Jane Smith,34
   3,Emily Davis,23
   4,Michael Brown,45
   5,Linda Johnson,30
4. Then perform the following to create tables:

Create Managed Table

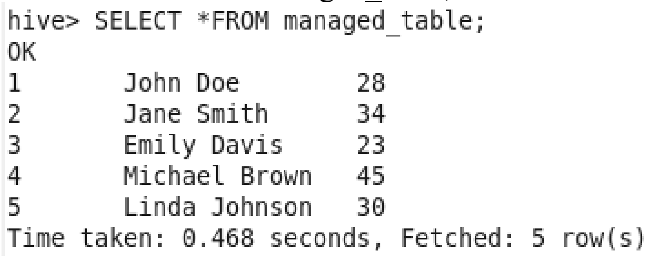
CREATE TABLE managed_table (id INT,name STRING,age INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;



```
grunt> grouped_data = GROUP formatted_data BY city;
2025-02-02 15:01:15,197 [main] WARN  org.apache.pig.PigServer - Encountered Warning IMPLICIT_CAST
_TO_FLOAT 1 time(s).
grunt> DUMP grouped_data;
```

LOAD DATA LOCAL INPATH '/home/cloudera/mydata.csv' INTO TABLE managed_table;

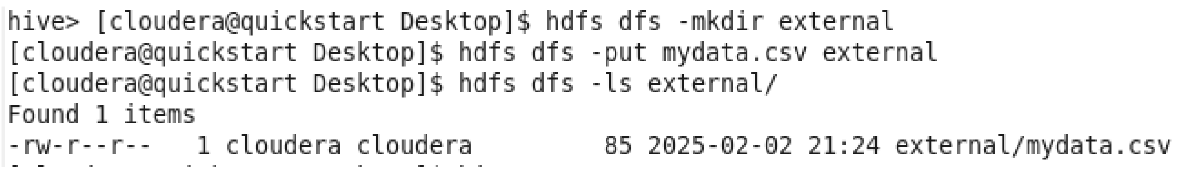
SELECT * FROM managed_table;



```
hive> SELECT *FROM managed_table;
OK
1       John Doe        28
2       Jane Smith      34
3       Emily Davis     23
4       Michael Brown   45
5       Linda Johnson   30
Time taken: 0.468 seconds, Fetched: 5 row(s)
```

Create External Table

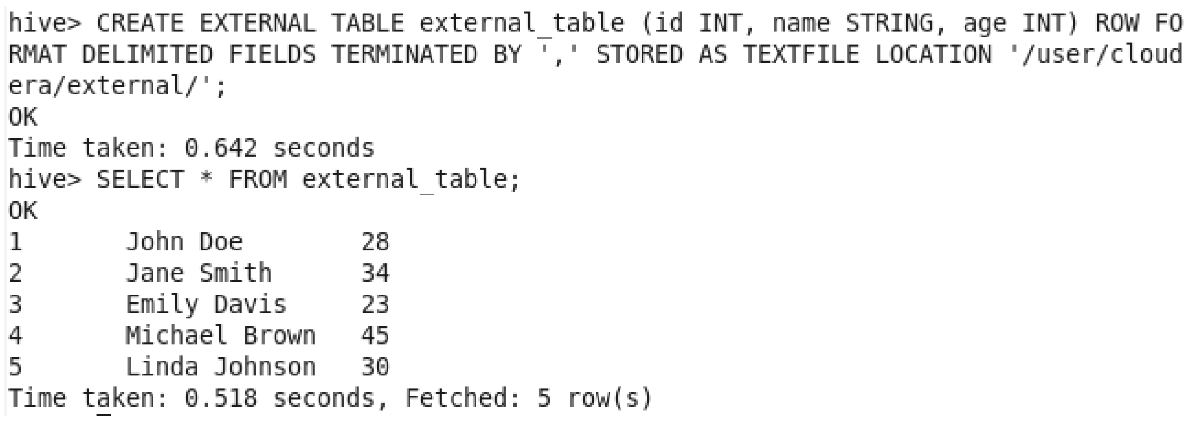
hdfs dfs -mkdir external
hdfs dfs -put mydata.csv external



```
hive> [cloudera@quickstart Desktop]$ hdfs dfs -mkdir external
[cloudera@quickstart Desktop]$ hdfs dfs -put mydata.csv external
[cloudera@quickstart Desktop]$ hdfs dfs -ls external/
Found 1 items
-rw-r--r--   1 cloudera cloudera         85 2025-02-02 21:24 external/mydata.csv
```

CREATE EXTERNAL TABLE external_table (id INT, name STRING, age INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/external/';

SELECT * FROM external_table;



```
hive> CREATE EXTERNAL TABLE external_table (id INT, name STRING, age INT) ROW FO
RMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/cloud
era/external/';
OK
Time taken: 0.642 seconds
hive> SELECT * FROM external_table;
OK
1       John Doe        28
2       Jane Smith      34
3       Emily Davis     23
4       Michael Brown   45
5       Linda Johnson   30
Time taken: 0.518 seconds, Fetched: 5 row(s)
```

Practical No. 6

Aim: Create a namespace in HBase with tables. Perform DML queries on the tables.

Explanation:
HBase is an open-source non-relational distributed database modeled after Google's Bigtable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System) or Alluxio, providing Bigtable-like capabilities for Hadoop.
That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).'

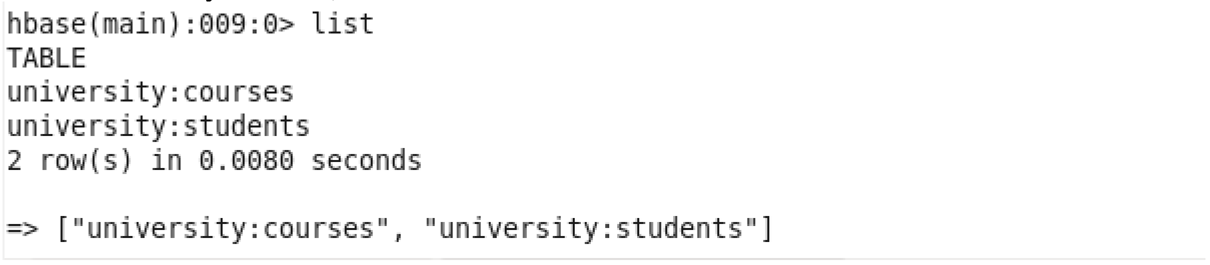
- **Start HADOOP, YARN, ZOOKEEPER and HBASE**

Create a namespace and Perform DML queries in table

```
create_namespace 'university'
```

Create Tables Under the Namespace

```
create 'university:students', 'info', 'marks'
create 'university:courses', 'details'
```



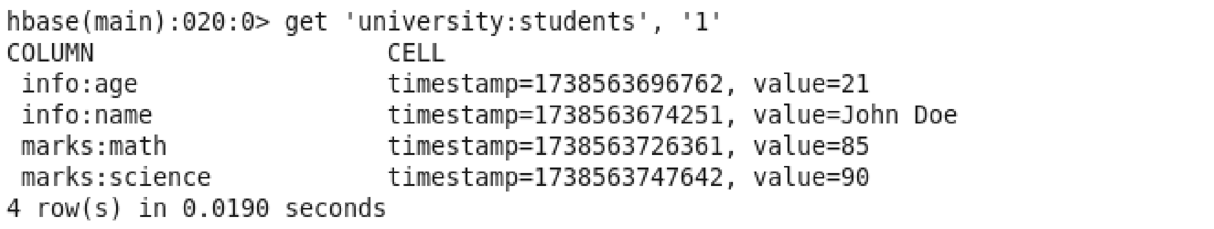
```
hbase(main):009:0> list
TABLE
university:courses
university:students
2 row(s) in 0.0080 seconds

=> ["university:courses", "university:students"]
```

Insert Data (Put Queries)

```
put 'university:students', '1', 'info:name', 'John Doe'
put 'university:students', '1', 'info:age', '21'
put 'university:students', '1', 'marks:math', '85'
put 'university:students', '1', 'marks:science', '90'

put 'university:students', '2', 'info:name', 'Jane Smith'
put 'university:students', '2', 'info:age', '22'
put 'university:students', '2', 'marks:math', '78'
put 'university:students', '2', 'marks:science', '88'

put 'university:courses', '101', 'details:name', 'Computer Science'
put 'university:courses', '101', 'details:credits', '4'
```

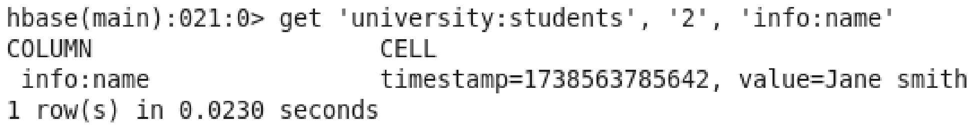
Retrieve Data (Get Queries)

```
get 'university:students', '1'
```



```
hbase(main):020:0> get 'university:students', '1'
COLUMN                    CELL
 info:age                 timestamp=1738563696762, value=21
 info:name                timestamp=1738563674251, value=John Doe
 marks:math               timestamp=1738563726361, value=85
 marks:science            timestamp=1738563747642, value=90
4 row(s) in 0.0190 seconds
```

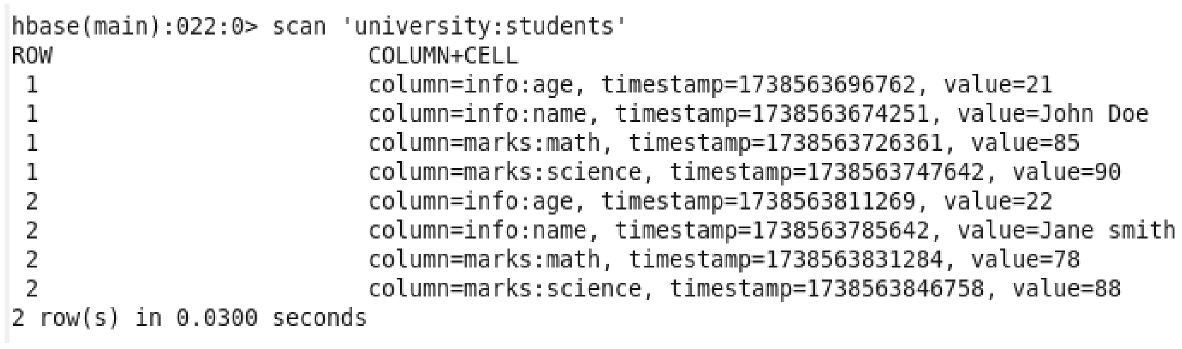
get 'university:students', '2', 'info:name'



```
hbase(main):021:0> get 'university:students', '2', 'info:name'
COLUMN                    CELL
 info:name                timestamp=1738563785642, value=Jane smith
1 row(s) in 0.0230 seconds
```

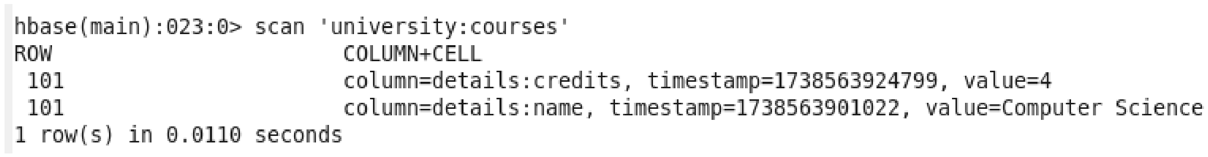
Scan Data (Scan Queries)

scan 'university:students'



```
hbase(main):022:0> scan 'university:students'
ROW                          COLUMN+CELL
 1                           column=info:age, timestamp=1738563696762, value=21
 1                           column=info:name, timestamp=1738563674251, value=John Doe
 1                           column=marks:math, timestamp=1738563726361, value=85
 1                           column=marks:science, timestamp=1738563747642, value=90
 2                           column=info:age, timestamp=1738563811269, value=22
 2                           column=info:name, timestamp=1738563785642, value=Jane smith
 2                           column=marks:math, timestamp=1738563831284, value=78
 2                           column=marks:science, timestamp=1738563846758, value=88
2 row(s) in 0.0300 seconds
```

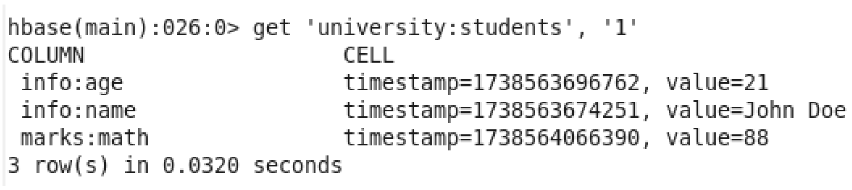
scan 'university:courses'



```
hbase(main):023:0> scan 'university:courses'
ROW                          COLUMN+CELL
 101                         column=details:credits, timestamp=1738563924799, value=4
 101                         column=details:name, timestamp=1738563901022, value=Computer Science
1 row(s) in 0.0110 seconds
```

Update Data (Another Put Query)

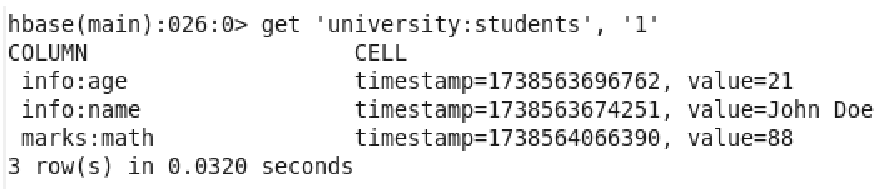
put 'university:students', '1', 'marks:math', '88'



```
hbase(main):026:0> get 'university:students', '1'
COLUMN                    CELL
 info:age                 timestamp=1738563696762, value=21
 info:name                timestamp=1738563674251, value=John Doe
 marks:math               timestamp=1738564066390, value=88
3 row(s) in 0.0320 seconds
```

Delete Specific Data (Delete Query)

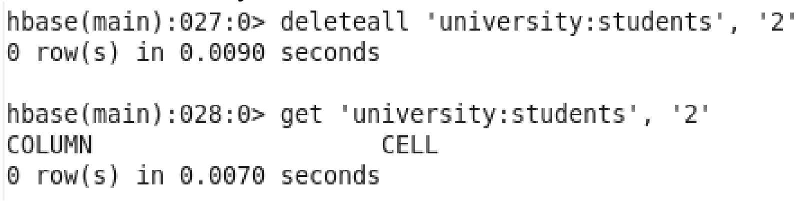
delete 'university:students', '1', 'marks:science'



```
hbase(main):026:0> get 'university:students', '1'
COLUMN                    CELL
 info:age                 timestamp=1738563696762, value=21
 info:name                timestamp=1738563674251, value=John Doe
 marks:math               timestamp=1738564066390, value=88
3 row(s) in 0.0320 seconds
```

<u>Delete an Entire Row</u>

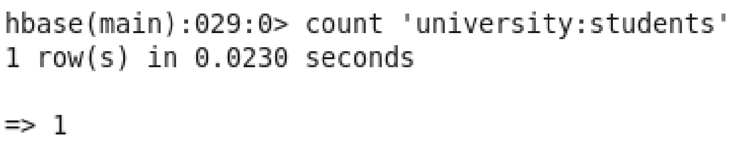
deleteall 'university:students', '2'



```
hbase(main):027:0> deleteall 'university:students', '2'
0 row(s) in 0.0090 seconds

hbase(main):028:0> get 'university:students', '2'
COLUMN                    CELL
0 row(s) in 0.0070 seconds
```

<u>Count the Number of Rows</u>

count 'university:students'

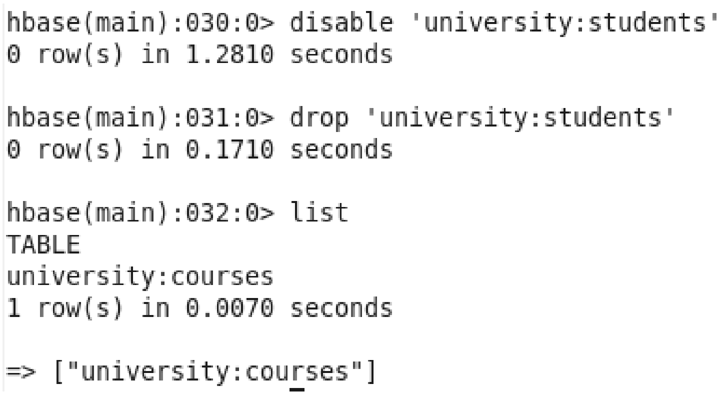


```
hbase(main):029:0> count 'university:students'
1 row(s) in 0.0230 seconds

=> 1
```

<u>Drop a Table</u>

disable 'university:students'

drop 'university:students'



```
hbase(main):030:0> disable 'university:students'
0 row(s) in 1.2810 seconds

hbase(main):031:0> drop 'university:students'
0 row(s) in 0.1710 seconds

hbase(main):032:0> list
TABLE
university:courses
1 row(s) in 0.0070 seconds

=> ["university:courses"]
```

Practical No. 7

Aim: Create a document and collection in MongoDB, performing advanced queries with conditional operators and aggregate functions

Explanation:

Creating Documents and Collections in MongoDB

1. Creating a Document: A document in MongoDB is a JSON-like structure used to store data. It consists of key-value pairs and supports nested structures. A typical document might look like:

```
{
"name": "Alice",
"age": 28,
"skills": ["JavaScript", "Python"],
"isEmployed": true
}
```

2. Creating a Collection: A collection in MongoDB is analogous to a table in relational databases, but it does not enforce a strict schema. Collections automatically get created when the first document is inserted.

Example:

```
db.users.insertOne({ name: "Alice", age: 28 });
```

Performing Advanced Queries with Conditional Operators

Conditional Operators:

MongoDB provides several conditional operators to refine query results.

Examples include:

$eq (equal to)

$ne (not equal to)

$gt (greater than), $lt (less than)

$in (matches any value in an array)

Example Query: Retrieve users aged above 25 but below 40.

```
db.users.find({ age: { $gt: 25, $lt: 40 } });
```

Logical Operators:

Combine multiple conditions with operators like $and, $or, $not.

Example: Find users aged above 30 or employed.

```
db.users.find({ $or: [{ age: { $gt: 30 } }, { isEmployed: true }] });
```

Using Aggregate Functions

MongoDB's aggregation framework processes data through a pipeline of stages, such as $match, $group, and $sort.

Example: Calculate average age by employment status.

```
db.users.aggregate([
{ $group: { _id: "$isEmployed", avgAge: { $avg: "$age" } } }
]);
```

Pipeline Stages:

1. $match: Filters documents based on criteria.

2. $group: Groups documents and performs operations like sum, avg, min, max.

3. $sort: Orders the results.

By combining these functionalities, MongoDB allows for powerful and flexible data management and analysis.

MongoDB Advanced Queries with Conditional Operators and Aggregate Functions

Step 1: Creating a MongoDB Collection and Document

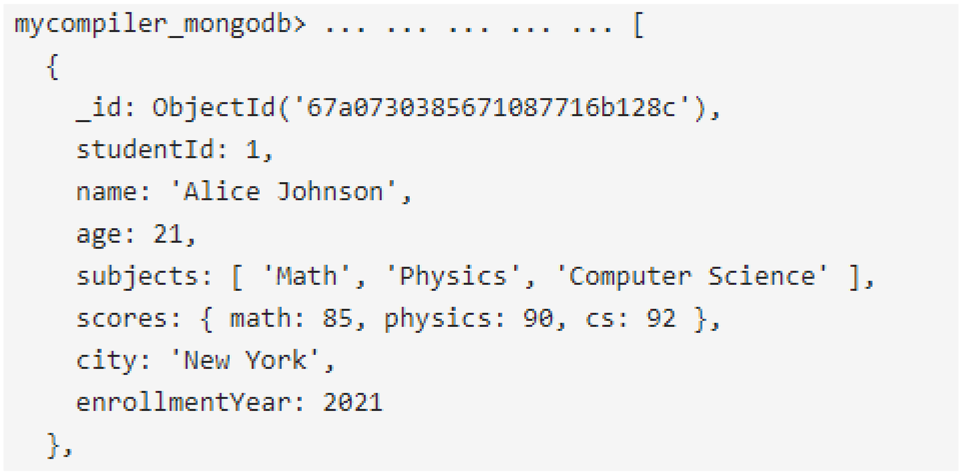
Create a collection student in MongoDB and insert data in it.

1. Insert documents into the students collection:

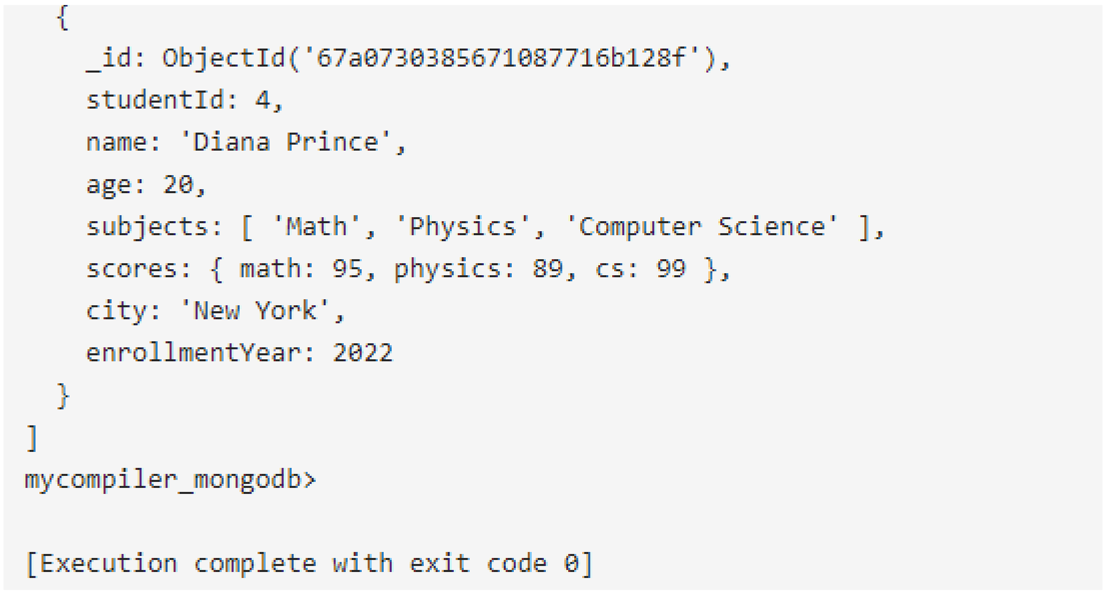
```
db.students.insertMany([
{
studentId: 1,
name: "Alice Johnson",
age: 21,
subjects: ["Math", "Physics", "Computer Science"],
scores: { math: 85, physics: 90, cs: 92 },
city: "New York",
enrollmentYear: 2021
},
{
studentId: 2,
name: "Bob Smith",
age: 22,
subjects: ["Math", "Biology", "Chemistry"],

scores: { math: 75, biology: 80, chemistry: 78 },
city: "San Francisco",
enrollmentYear: 2020
},
{
studentId: 3,
name: "Charlie Brown",
age: 23,
subjects: ["Math", "Physics", "Economics"],
scores: { math: 65, physics: 70, economics: 75 },
city: "Los Angeles",
enrollmentYear: 2019
},
{
studentId: 4,
name: "Diana Prince",
age: 20,
subjects: ["Math", "Physics", "Computer Science"],
scores: { math: 95, physics: 89, cs: 99 },
city: "New York",
enrollmentYear: 2022
}
]);
```

Step 2: Performing Advanced Queries
1. Using Conditional Operators
Find students who scored above 80 in Math and are enrolled after 2020.

```
db.students.find({
$and: [
{ "scores.math": { $gt: 80 } },
{ enrollmentYear: { $gt: 2020 } }
]
});
```



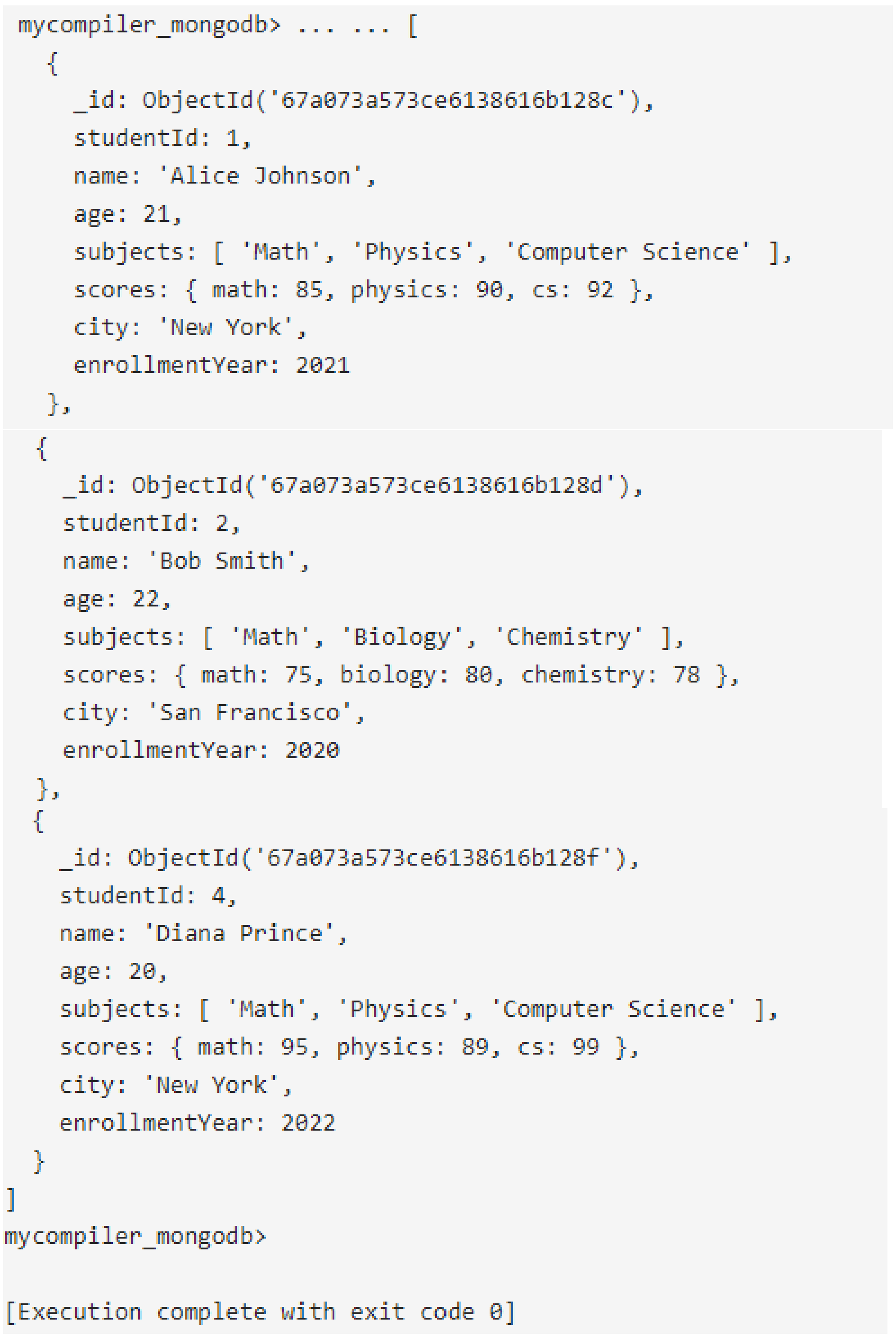
```
mycompiler_mongodb> ... ... ... ... ... [
  {
    _id: ObjectId('67a0730385671087716b128c'),
    studentId: 1,
    name: 'Alice Johnson',
    age: 21,
    subjects: [ 'Math', 'Physics', 'Computer Science' ],
    scores: { math: 85, physics: 90, cs: 92 },
    city: 'New York',
    enrollmentYear: 2021
  },
```



```
  {
    _id: ObjectId('67a0730385671087716b128f'),
    studentId: 4,
    name: 'Diana Prince',
    age: 20,
    subjects: [ 'Math', 'Physics', 'Computer Science' ],
    scores: { math: 95, physics: 89, cs: 99 },
    city: 'New York',
    enrollmentYear: 2022
  }
]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

2. Query with $or
Find students from either "New York" or "San Francisco".

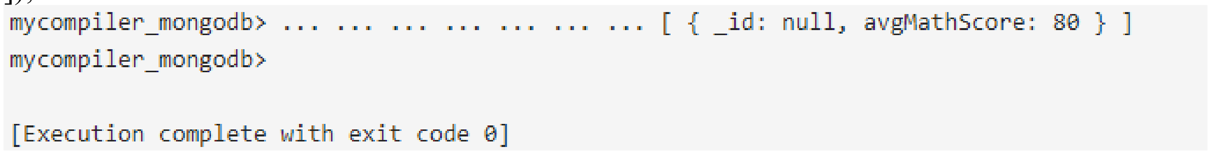
```
db.students.find({
city: { $in: ["New York", "San Francisco"] }
});
```



```
mycompiler_mongodb> ... ... [
  {
    _id: ObjectId('67a073a573ce6138616b128c'),
    studentId: 1,
    name: 'Alice Johnson',
    age: 21,
    subjects: [ 'Math', 'Physics', 'Computer Science' ],
    scores: { math: 85, physics: 90, cs: 92 },
    city: 'New York',
    enrollmentYear: 2021
  },
  {
    _id: ObjectId('67a073a573ce6138616b128d'),
    studentId: 2,
    name: 'Bob Smith',
    age: 22,
    subjects: [ 'Math', 'Biology', 'Chemistry' ],
    scores: { math: 75, biology: 80, chemistry: 78 },
    city: 'San Francisco',
    enrollmentYear: 2020
  },
  {
    _id: ObjectId('67a073a573ce6138616b128f'),
    studentId: 4,
    name: 'Diana Prince',
    age: 20,
    subjects: [ 'Math', 'Physics', 'Computer Science' ],
    scores: { math: 95, physics: 89, cs: 99 },
    city: 'New York',
    enrollmentYear: 2022
  }
]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

3. Aggregation with Aggregate Functions
a. Calculate the Average Math Score

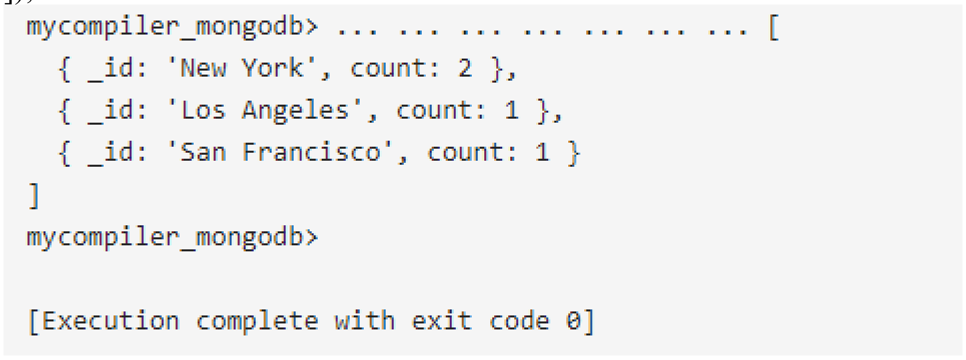
```
db.students.aggregate([
{
$group: {
_id: null,
avgMathScore: { $avg: "$scores.math" }
}
}
]);
```



```
mycompiler_mongodb> ... ... ... ... ... ... ... [ { _id: null, avgMathScore: 80 } ]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

b. Count Students Per City

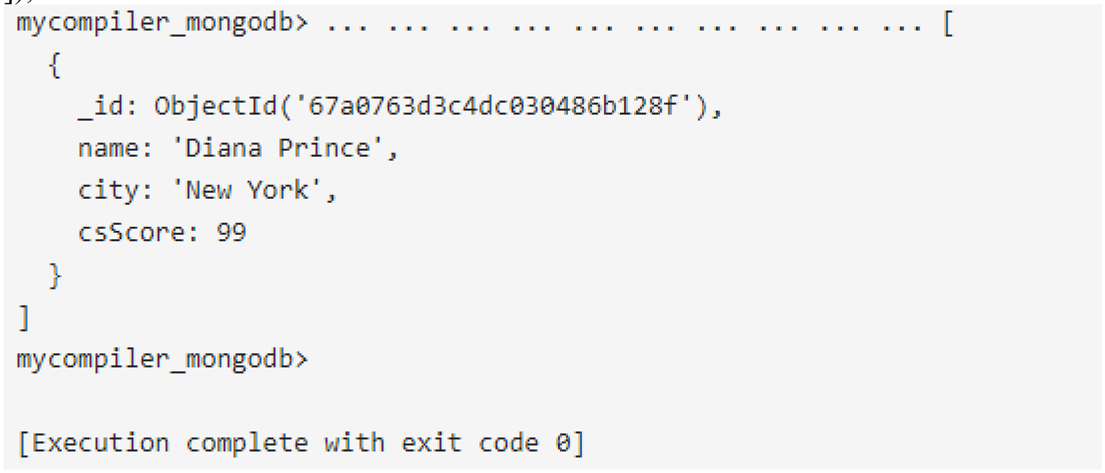
```
db.students.aggregate([
{
$group: {
_id: "$city",
count: { $sum: 1 }
}
}
]);
```



```
mycompiler_mongodb> ... ... ... ... ... ... ... [
  { _id: 'New York', count: 2 },
  { _id: 'Los Angeles', count: 1 },
  { _id: 'San Francisco', count: 1 }
]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

c. Find the Top Scorer in Computer Science

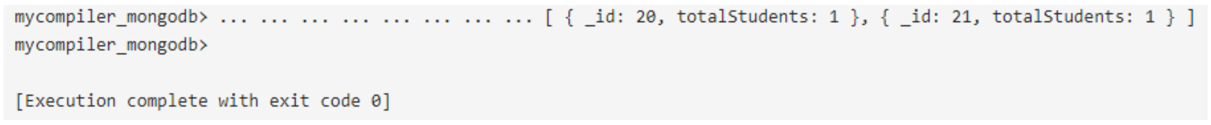
```
db.students.aggregate([
{
$project: {
name: 1,
city: 1,
csScore: "$scores.cs"

}
},
{ $sort: { csScore: -1 } },
{ $limit: 1 }
]);
```



```
mycompiler_mongodb> ... ... ... ... ... ... ... ... ... ... [
  {
    _id: ObjectId('67a0763d3c4dc030486b128f'),
    name: 'Diana Prince',
    city: 'New York',
    csScore: 99
  }
]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

d. Filter and Group Students Enrolled After 2020 by Age

```
db.students.aggregate([
{ $match: { enrollmentYear: { $gt: 2020 } } },
{
$group: {
_id: "$age",
totalStudents: { $sum: 1 }
}
}
]);
```



```
mycompiler_mongodb> ... ... ... ... ... ... ... ... [ { _id: 20, totalStudents: 1 }, { _id: 21, totalStudents: 1 } ]
mycompiler_mongodb>

[Execution complete with exit code 0]
```

Practical No. 8

Aim: Perform aggregation using Map() and Reduce() functions in MongoDB
Explanation:
In MongoDB, Map-Reduce is a data processing paradigm used to perform large-scale data aggregation. It involves two main steps:

- Map Function: The map() function is applied to each document in the input collection.
- It emits key-value pairs based on the logic defined in the function.
- Example: For each document, emit a key (e.g., a category) and a value (e.g., a sales amount).
-
- Reduce Function: The reduce() function takes the emitted keys and aggregates their corresponding values.
- It is used to combine and summarize the data.
- Example: Sum up all sales amounts for each category.
-

Steps in Map-Reduce:
1. Mapping: Documents are processed, and key-value pairs are emitted.
2. Shuffling: Intermediate key-value pairs are grouped by keys.
3. Reducing: Aggregated results are computed using the reduce function.

MongoDB Command to Create and Insert Data
```
db.sales.insertMany([
{ "_id": 1, "product": "Laptop", "category": "Electronics", "amount": 1500 },
{ "_id": 2, "product": "Headphones", "category": "Electronics", "amount": 200 },
{ "_id": 3, "product": "Shirt", "category": "Clothing", "amount": 50 },
{ "_id": 4, "product": "Pants", "category": "Clothing", "amount": 100 },
{ "_id": 5, "product": "Phone", "category": "Electronics", "amount": 800 },
{ "_id": 6, "product": "Shoes", "category": "Footwear", "amount": 120 }
]);
```
Map-Reduce Implementation
1. Map Function

The map function emits a key-value pair where the key is the category and the value is the sales amount.
```
var mapFunction = function() {
emit(this.category, this.amount);
};
```
1. Reduce Function

The reduce function aggregates the values (sales amounts) for each key (category).

```
var reduceFunction = function(key, values) {
return Array.sum(values);
};
```

1. **Executing Map-Reduce**

```
db.sales.mapReduce(
mapFunction,
reduceFunction,
{
out: "category_totals"
}
);
```

```
db.category_totals.find()
```

```
{ "_id": "Electronics", "value": 2500 }
{ "_id": "Clothing", "value": 150 }
{ "_id": "Footwear", "value": 120 }
```

1. **Result**

After running the above, the results will be stored in a new collection named category_totals.
To view the results:
db.category_totals.find().pretty();

```
{
    "_id" : "Electronics",
    "value" : 2500
}
{
    "_id" : "Clothing",
    "value" : 150
}
{
    "_id" : "Footwear",
    "value" : 120
}
```

Practical No. 9

Aim: Creating nodes and relationships in graph database using cypher. Perform exploration queries

Guide to Creating Nodes, Relationships, and Exploring a Graph Database with Cypher

1. Setting up the Graph Database

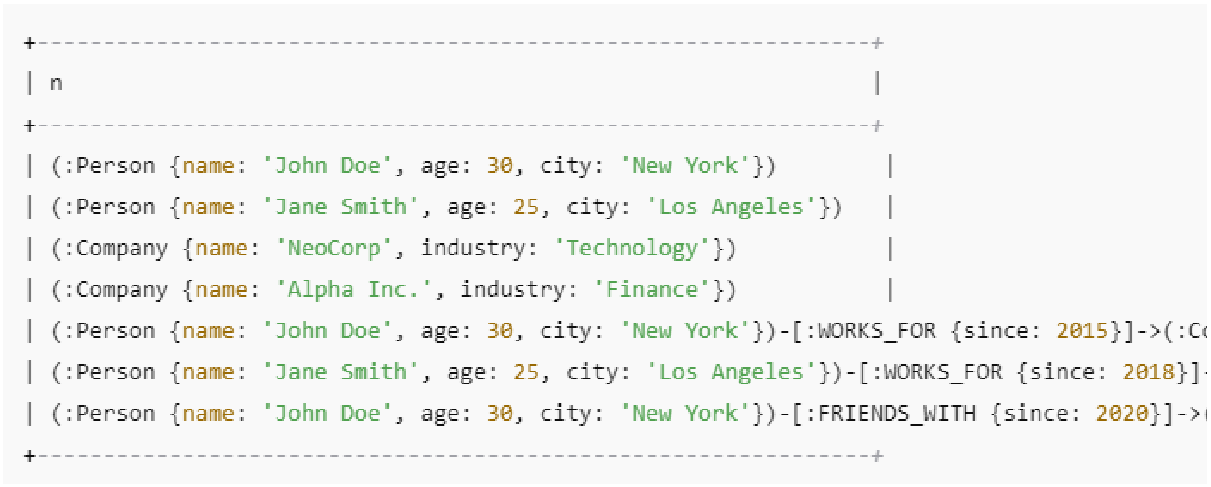
We'll use Neo4j for this example. You can either:
- Download and install Neo4j Desktop.
- Use the Neo4j Aura Cloud free tier.

2. Creating Nodes

Nodes are entities in a graph database. Each node can have:
- A label (to categorize nodes).
- Properties (key-value pairs for details).

Example: Creating Nodes

```
CREATE
(john:Person {name: 'John Doe', age: 30, city: 'New York'}),
(jane:Person {name: 'Jane Smith', age: 25, city: 'Los Angeles'}),
(neoCorp:Company {name: 'NeoCorp', industry: 'Technology'}),
(alphaInc:Company {name: 'Alpha Inc.', industry: 'Finance'});
MATCH (n) RETURN n;
```



```
+-----------------------------------------------------------------+
| n                                                               |
+-----------------------------------------------------------------+
| (:Person {name: 'John Doe', age: 30, city: 'New York'})         |
| (:Person {name: 'Jane Smith', age: 25, city: 'Los Angeles'})    |
| (:Company {name: 'NeoCorp', industry: 'Technology'})            |
| (:Company {name: 'Alpha Inc.', industry: 'Finance'})            |
| (:Person {name: 'John Doe', age: 30, city: 'New York'})-[:WORKS_FOR {since: 2015}]->(:C
| (:Person {name: 'Jane Smith', age: 25, city: 'Los Angeles'})-[:WORKS_FOR {since: 2018}]-
| (:Person {name: 'John Doe', age: 30, city: 'New York'})-[:FRIENDS_WITH {since: 2020}]->
+-----------------------------------------------------------------+
```

3. Creating Relationships

Relationships connect nodes and can also have properties.

Example: Creating Relationships

```
MATCH (john:Person {name: 'John Doe'}), (jane:Person {name: 'Jane Smith'}),
(neoCorp:Company {name: 'NeoCorp'}), (alphaInc:Company {name: 'Alpha
Inc.'})
CREATE
(john)-[:WORKS_FOR {since: 2015}]->(neoCorp),
(jane)-[:WORKS_FOR {since: 2018}]->(alphaInc),
(john)-[:FRIENDS_WITH {since: 2020}]->(jane);
```

```
+-----------------------------------------+
| p.name          | p.age | p.city       |
+-----------------------------------------+
| 'John Doe'    | 30     | 'New York'   |
| 'Jane Smith' | 25     | 'Los Angeles'|
+-----------------------------------------+
```

4. Performing Exploration Queries
4.1 Retrieve All Nodes
MATCH (n) RETURN n;

```
+--------------------------------------------+
| From             | Relationship  | To         |
+--------------------------------------------+
| 'John Doe'      | 'WORKS_FOR'    | 'NeoCorp'|
| 'Jane Smith'  | 'WORKS_FOR'    | 'Alpha Inc.' |
| 'John Doe'      | 'FRIENDS_WITH'| 'Jane Smith' |
+--------------------------------------------+
```

4.2 Retrieve Specific Nodes
MATCH (p:Person) RETURN p.name, p.age, p.city;

```
+-------------------------------------+
| Employee       | Company            |
+-------------------------------------+
| 'John Doe'    | 'NeoCorp'         |
+-------------------------------------+
```

4.3 Retrieve Relationships
MATCH (a)-[r]->(b) RETURN a.name AS From, TYPE(r) AS Relationship, b.name AS To;

```
+--------------------+
| MutualFriend       |
+--------------------+
| 'Jane Smith'       |
+--------------------+
```

4.4 Filter by Property
MATCH (p:Person)-[:WORKS_FOR]->(c:Company {name: 'NeoCorp'})
RETURN p.name AS Employee, c.name AS Company;

```
+--------------------------+
| Company      | Employees   |
+--------------------------+
| 'NeoCorp'   | ['John Doe']|
| 'Alpha Inc.' | ['Jane Smith'] |
+--------------------------+
```

4.5 Find Mutual Relationships
MATCH (p1:Person {name: 'John Doe'})-[:FRIENDS_WITH]-(mutual)-[:FRIENDS_WITH]-(p2:Person {name: 'Jane Smith'})
RETURN mutual.name AS MutualFriend;

```
+-------------------------------------------------------+
| Person       | SuggestedFriend | CommonCompany         |
+-------------------------------------------------------+
| 'John Doe'   | 'Alice Johnson' | 'NeoCorp'             |
| 'Jane Smith'| 'Bob Williams'  | 'Alpha Inc.'          |
+-------------------------------------------------------+
```

4.6 Find Employees of Companies
MATCH (p:Person)-[:WORKS_FOR]->(c:Company)
RETURN c.name AS Company, COLLECT(p.name) AS Employees;

```
+-------------------------------------------------------------+
| path                                                        |
+-------------------------------------------------------------+
| (:Person {name: 'John Doe'})-[:FRIENDS_WITH {since: 2020}]->(:Person {name: 'Jane Smith'
+-------------------------------------------------------------+
```

4.7 Suggest Connections
MATCH
(p1:Person)-[:WORKS_FOR]->(c:Company)<-[:WORKS_FOR]-(p2:Person)
WHERE NOT (p1)-[:FRIENDS_WITH]-(p2) AND p1 <> p2
RETURN p1.name AS Person, p2.name AS SuggestedFriend, c.name AS CommonCompany;

```
+--------------------------------------+
| Person        | NumberOfConnections    |
+--------------------------------------+
| 'John Doe'    | 1                      |
| 'Jane Smith' | 1                      |
+--------------------------------------+
```

4.8 Shortest Path
MATCH path = shortestPath((john:Person {name: 'John Doe'})-[:FRIENDS_WITH*..5]-(jane:Person {name: 'Jane Smith'}))
RETURN path;

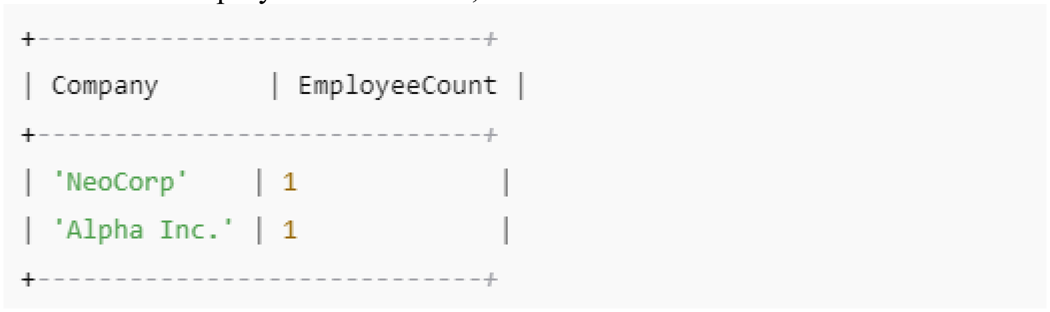
5. Advanced Exploration

5.1 Analyze Connectivity

MATCH (p:Person)-[:FRIENDS_WITH]-(friend)
RETURN p.name AS Person, COUNT(friend) AS NumberOfConnections;

5.2 Find Influential Companies

MATCH (c:Company)<-[:WORKS_FOR]-(employee)
RETURN c.name AS Company, COUNT(employee) AS EmployeeCount
ORDER BY EmployeeCount DESC;



```
+----------------------------+
| Company       | EmployeeCount |
+----------------------------+
| 'NeoCorp'     | 1             |
| 'Alpha Inc.'  | 1             |
+----------------------------+
```

6. Cleaning Up the Database

To delete all data:
MATCH (n) DETACH DELETE n;