Exp No: 6

Handling JSON data using HDFS and Python AIM:

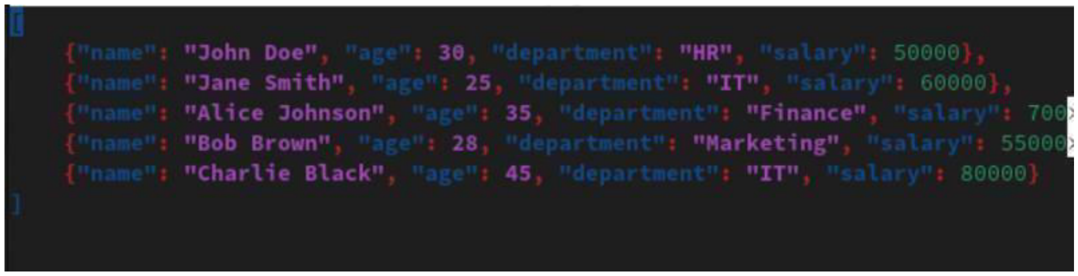
To handle JSON data using HDFS and python.

PROCEDURE:

Step 1: Create json file on bash & save as emp.json

nano emp.json ; Paste the below content on it

```
[
    {"name": "John Doe", "age": 30, "department": "HR", "salary": 50000},
    {"name": "Jane Smith", "age": 25, "department": "IT", "salary": 60000},
    {"name": "Alice Johnson", "age": 35, "department": "Finance", "salary": 70000},
    {"name": "Bob Brown", "age": 28, "department": "Marketing", "salary": 55000},
{"name": "Charlie Black", "age": 45, "department": "IT", "salary": 80000}
]
```

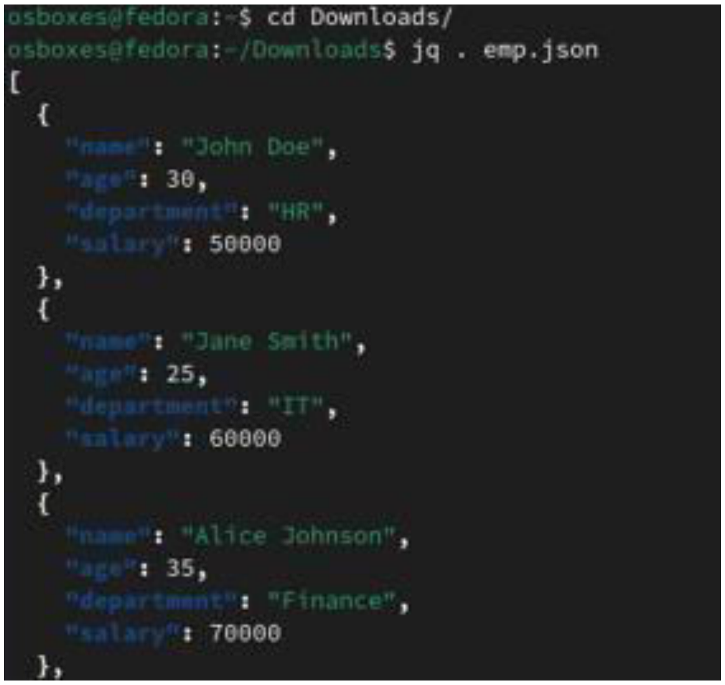


```
[
    {"name": "John Doe", "age": 30, "department": "HR", "salary": 50000},
    {"name": "Jane Smith", "age": 25, "department": "IT", "salary": 60000},
    {"name": "Alice Johnson", "age": 35, "department": "Finance", "salary": 700
    {"name": "Bob Brown", "age": 28, "department": "Marketing", "salary": 55000
    {"name": "Charlie Black", "age": 45, "department": "IT", "salary": 80000}
]
```

Step 2 : Check json is readable or any error by giving

install jq by sudo apt-get install jq hadoop@Ubuntu:~$

jq . emp.json



```
osboxes@fedora:~$ cd Downloads/
osboxes@fedora:~/Downloads$ jq . emp.json
[
  {
    "name": "John Doe",
    "age": 30,
    "department": "HR",
    "salary": 50000
  },
  {
    "name": "Jane Smith",
    "age": 25,
    "department": "IT",
    "salary": 60000
  },
  {
    "name": "Alice Johnson",
    "age": 35,
    "department": "Finance",
    "salary": 70000
  },
```

Step 3: Install pandas and hdfs dependencies for

python. Step 4: Create process_data.py file from hdfs

import InsecureClient import pandas as pd import json

# Connect to HDFS hdfs_client =

InsecureClient('http://localhost:9870', user='hdfs')

# Read JSON data from HDFS try: with

hdfs_client.read('/home/hadoop/emp.json', encoding='utf-8') as reader:

json_data = reader.read() # Read the raw data as a string if not json_data.strip(): #

Check if data is empty raise ValueError("The JSON file is empty.") print(f"Raw

JSON Data: {json_data[:1000]}") # Print first 1000 characters for debugging data =

json.loads(json_data) # Load the JSON data except json.JSONDecodeError as e:

print(f"JSON Decode Error: {e}")

exit(1) except Exception as e: print(f"Error

reading or parsing JSON data: {e}") exit(1)

```
# Convert JSON data to DataFrame try:      df =
pd.DataFrame(data) except ValueError as e:
print(f"Error converting JSON data to DataFrame: {e}")
    exit(1)

# Projection: Select only 'name' and 'salary' columns projected_df
= df[['name', 'salary']]

# Aggregation: Calculate total salary total_salary
= df['salary'].sum()

# Count: Number of employees earning more than 50000 high_earners_count
= df[df['salary'] > 50000].shape[0]

# Limit: Get the top 5 highest earners top_5_earners
= df.nlargest(5, 'salary')

# Skip: Skip the first 2 employees skipped_df
= df.iloc[2:]

# Remove: Remove employees from a specific department filtered_df
= df[df['department'] != 'IT']

# Save the filtered result back to HDFS filtered_json = filtered_df.to_json(orient='records') try:      with
hdfs_client.write('/home/hadoop/filtered_employees.json', encoding='utf-8', overwrite=True) as writer:
```

writer.write(filtered_json) print("Filtered JSON file saved successfully.") except Exception as e: print(f"Error saving filtered JSON data: {e}") exit(1)

# Print results print(f"Projection: Select only name and salary columns") print(f"{projected_df}")

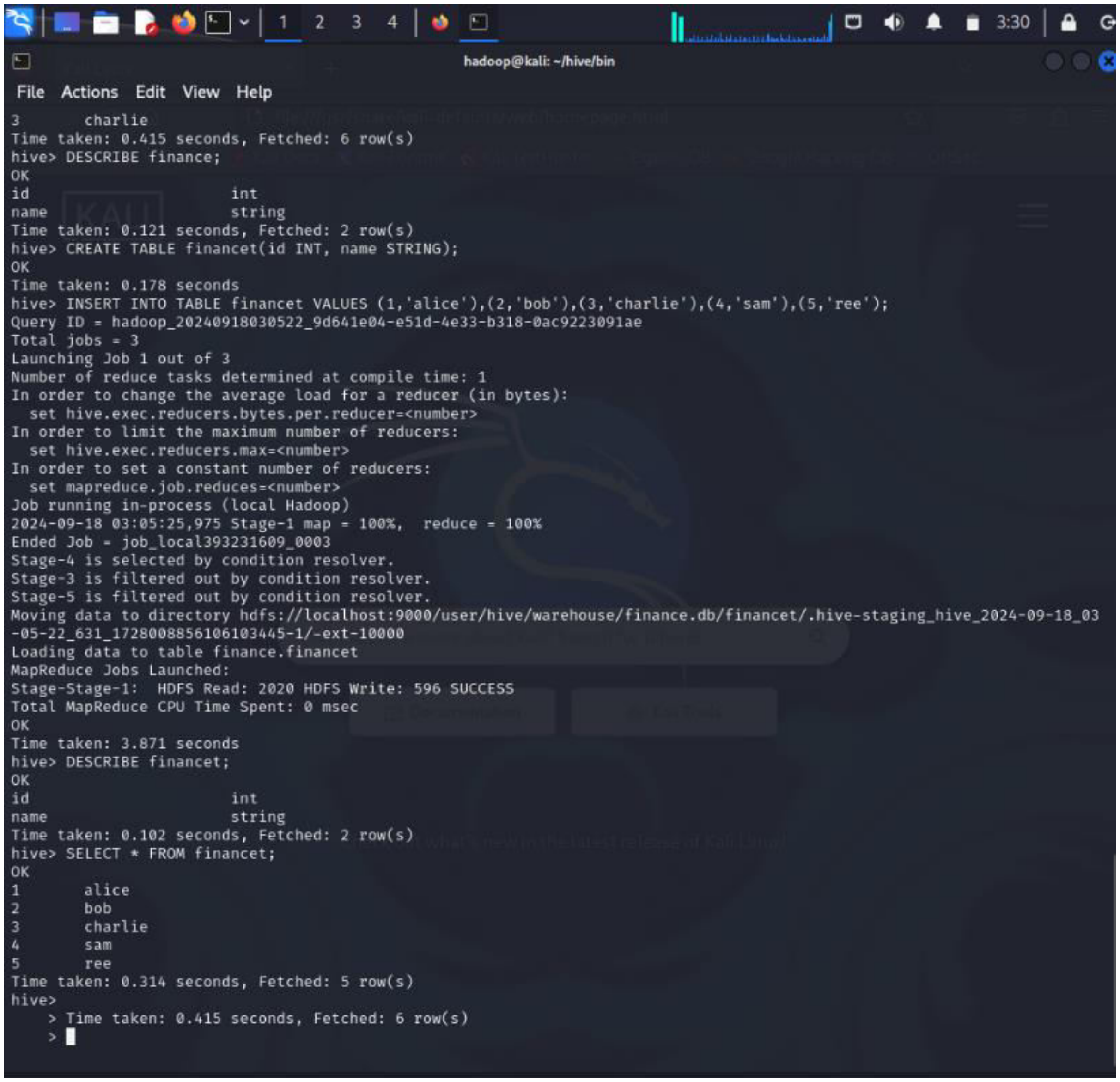
print(f"Aggregation: Calculate total salary")

print(f"Total Salary: {total_salary}") print(f"\n")

print(f"# Count: Number of employees earning more than 50000")

print(f"Number of High Earners (>50000): {high_earners_count}") print(f"\n") print(f"limit Top 5 highest salary")

print(f"Top 5 Earners: \n{top_5_earners}") print(f"\n") print(f"Skipped DataFrame (First 2 rows skipped): \n{skipped_df}") print(f"\n") print(f"Filtered DataFrame (Sales department removed): \n{filtered_df}") **Step 5: run the file by** bash: python3 process_data.py



```
3       charlie
Time taken: 0.415 seconds, Fetched: 6 row(s)
hive> DESCRIBE finance;
OK
id                      int
name                    string
Time taken: 0.121 seconds, Fetched: 2 row(s)
hive> CREATE TABLE financet(id INT, name STRING);
OK
Time taken: 0.178 seconds
hive> INSERT INTO TABLE financet VALUES (1,'alice'),(2,'bob'),(3,'charlie'),(4,'sam'),(5,'ree');
Query ID = hadoop_20240918030522_9d641e04-e51d-4e33-b318-0ac9223091ae
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2024-09-18 03:05:25,975 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local393231609_0003
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/finance.db/financet/.hive-staging_hive_2024-09-18_03
-05-22_631_1728008856106103445-1/-ext-10000
Loading data to table finance.financet
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 2020 HDFS Write: 596 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 3.871 seconds
hive> DESCRIBE financet;
OK
id                      int
name                    string
Time taken: 0.102 seconds, Fetched: 2 row(s)
hive> SELECT * FROM financet;
OK
1       alice
2       bob
3       charlie
4       sam
5       ree
Time taken: 0.314 seconds, Fetched: 5 row(s)
hive>
    > Time taken: 0.415 seconds, Fetched: 6 row(s)
    >
```

RESULT:

Experiment has been successfully executed and output has been verified.