Big Data Analytics
Lab Codes

Execution Report

Week4 - Week10

Ashrut Alok Arora
210968206
DSE-A
Batch-2

Week4 (Pig Script)

Q1)
Consider a normal text file to learn the pig running modes and execution modes. Run the program locally and test it on Hadoop.

Ans)

input.txt

```
Hello world
This is a sample input file
It contains some text for testing purposes
```

script.pig

```
-- Example Pig script (script.pig)
-- Calculate the total count of words in a text file

-- Load input data from a text file
data = LOAD 'input.txt' USING PigStorage() AS
(line:chararray);

-- Tokenize each line into words
words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS
word;

-- Group and count the words
word_count = GROUP words BY word;
word_count_total = FOREACH word_count GENERATE group,
COUNT(words);

-- Store the result
STORE word_count_total INTO 'output' USING PigStorage();
```

Output

```
(a,1)
(It,1)
(is,1)
(for,1)
(This,1)
(file,1)
(some,1)
(text,1)
(Hello,1)
(input,1)
(world,1)
(sample,1)
(testing,1)
(contains,1)
(purposes,1)
```

Q2)
Write a pig program to count the number of word occurrences using python in different modes (local mode, MapReduce mode)

Ans)

Input.txt

```
Hello, this is a sample input text file.
It contains some words that will be counted.
Each word may appear multiple times in this file.
The purpose is to demonstrate word counting using Pig.
Pig is a tool used for big data processing.
```

script.pig

```
-- Pig script to count word occurrences using Python in
```

```
different modes

-- Load input data from a text file in HDFS
data = LOAD '/path/to/your/input.txt' USING PigStorage()
AS (line:chararray);

-- Tokenize each line into words
words = FOREACH data GENERATE FLATTEN(TOKENIZE(line)) AS
word;

-- Group and count the words
word_count = GROUP words BY word;
word_count_total = FOREACH word_count GENERATE group AS
word, COUNT(words) AS count;

-- Display the result on screen
DUMP word_count_total;
```

Output

```
(a,2)
(It,1)
(be,1)
(in,1)
(is,3)
(to,1)
(Pig,1)
(The,1)
(big,1)
(for,1)
(may,1)
(Each,1)
(Pig.,1)
(data,1)
(some,1)
(text,1)
```

```
(that,1)
(this,2)
(tool,1)
(used,1)
(will,1)
(word,2)
(Hello,1)
(file.,2)
(input,1)
(times,1)
(using,1)
(words,1)
(appear,1)
(sample,1)
(purpose,1)
(contains,1)
(counted.,1)
(counting,1)
(multiple,1)
(demonstrate,1)
(processing.,1)
(,0)
```

Q3)
Execute the pig script to find the “most popular moive in the dataset”. In this example we will be dealing with 2 files (ratings.data and movies.item).

Ans)

Code

```
-- Load ratings data
ratings = LOAD 'ratings.data' USING PigStorage('\t') AS
(userID:int, movieID:int, rating:int, timestamp:int);

-- Load movies data
movies = LOAD 'movies.item' USING PigStorage('|') AS
(movieID:int, title:chararray, releaseDate:chararray,
imdbLink:chararray);

-- Join ratings and movies on movieID
joined_data = JOIN ratings BY movieID, movies BY movieID;

-- Group by movie title and calculate total ratings
grouped_data = GROUP joined_data BY movies::title;
rating_counts = FOREACH grouped_data GENERATE group AS
title, COUNT(joined_data) AS total_ratings;

-- Find the movie with the highest number of ratings
sorted_ratings = ORDER rating_counts BY total_ratings
DESC;
top_movie = LIMIT sorted_ratings 1;

-- Display the result
DUMP top_movie;
```

Output

```
(Star Wars (1977),584)
```

Week5 (Spark)

Q1)
Create the dataset of your choice and perform word count program using spark tool.

Ans)
Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col

# Create a SparkSession
spark = SparkSession.builder \
    .appName("WordCount") \
    .getOrCreate()

# Create a DataFrame with sample text data
data = [("Hello world",),
        ("Hello Spark",),
        ("Spark is awesome",),
        ("World of Spark",)]
df = spark.createDataFrame(data, ["text"])

# Split the text into words
words = df.select(explode(split(col("text"), "
")).alias("word"))

# Perform word count
word_count =
words.groupBy("word").count().orderBy("count",
ascending=False)

# Show the result
```

```
word_count.show()

# Stop the SparkSession
spark.stop()
```

Output

```
+-------------+-----+
|         word|count|
+-------------+-----+
|      Spark's|    2|
|        world|    2|
|       Spark.|    2|
|        Hello|    2|
|       Spark |    1|
|      is     |    1|
|      awesome|    1|
+-------------+-----+
```

Q2)
Given a dataset of employee records containing (name, age, salary), use map transformation to transform each record into a tuple of (name, age * 2, salary)?

Reg.No	EmpName	Age	Salary
24	John	26	30000
34	Jack	40	80000
61	Joshi	25	35000
45	Jash	35	75000
34	Yash	40	60000
67	Smith	20	24000
42	Lion	42	56000
62	kate	50	76000
21	cassy	51	40000
10	ronald	57	65000
24	John	26	30000
67	Smith	20	24000
45	Jash	35	75000
21	cassy	51	40000

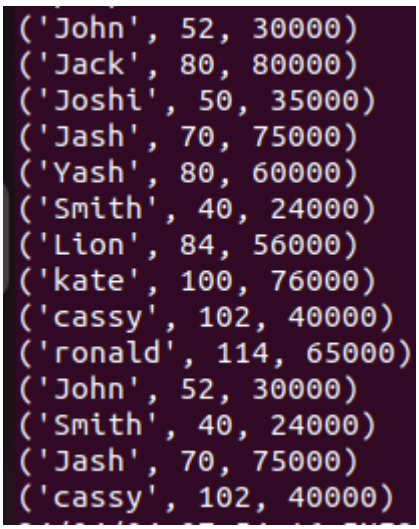
Ans)

Code

```
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("EmployeeRecords") \
    .getOrCreate()

# Define the dataset
data = [
    ("John", 26, 30000),
    ("Jack", 40, 80000),
    ("Joshi", 25, 35000),
    ("Jash", 35, 75000),
```

```
    ("Yash", 40, 60000),
    ("Smith", 20, 24000),
    ("Lion", 42, 56000),
    ("kate", 50, 76000),
    ("cassy", 51, 40000),
    ("ronald", 57, 65000),
    ("John", 26, 30000),
    ("Smith", 20, 24000),
    ("Jash", 35, 75000),
    ("cassy", 51, 40000)
]

# Create a DataFrame from the data
df = spark.createDataFrame(data, ["name", "age",
"salary"])

# Apply map transformation to transform each record
transformed_rdd = df.rdd.map(lambda row: (row.name,
row.age * 2, row.salary))

# Display the transformed records
transformed_records = transformed_rdd.collect()
for record in transformed_records:
    print(record)

# Stop the SparkSession
spark.stop()
```

Output



Q3)
From the same employee dataset, filter out employees whose salary is greater than 50000 using the filter transformation.

Ans)
Code

```
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("EmployeeSalaryFilter") \
    .getOrCreate()

# Define the employee dataset
data = [
    ("John", 26, 30000),
    ("Jack", 40, 80000),
    ("Joshi", 25, 35000),
    ("Jash", 35, 75000),
    ("Yash", 40, 60000),
    ("Smith", 20, 24000),
    ("Lion", 42, 56000),
```

```
    ("Kate", 50, 76000),
    ("Cassy", 51, 40000),
    ("Ronald", 57, 65000),
    ("John", 26, 30000),
    ("Smith", 20, 24000),
    ("Jash", 35, 75000),
    ("Cassy", 51, 40000)
]

# Create a DataFrame from the data
df = spark.createDataFrame(data, ["EmpName", "Age",
"Salary"])

# Apply filter transformation to filter out employees
with salary greater than 50000
filtered_df = df.filter(df["Salary"] > 50000)

# Show the filtered DataFrame
filtered_df.show()

# Stop the SparkSession
spark.stop()
```

Output

```
+-------+---+------+
|EmpName|Age|Salary|
+-------+---+------+
|   Jack| 40| 80000|
|   Jash| 35| 75000|
|   Yash| 40| 60000|
|   Lion| 42| 56000|
|   Kate| 50| 76000|
| Ronald| 57| 65000|
|   Jash| 35| 75000|
+-------+---+------+
```

Q4)
Create a text file that will have few sentences, use flatMap transformation to split each sentence into words.

Ans)

sentences.txt

```
Spark is a powerful distributed computing framework.
Python is widely used for data analysis and machine learning.
Big Data technologies are revolutionizing industries.
Machine learning algorithms help in predictive analytics.
Data scientists analyze large datasets to extract insights.
Apache Hadoop is a popular framework for distributed storage
and processing.
Deep learning models require large amounts of labeled data.
Natural language processing enables computers to understand
human language.
```

Code

```
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("SentenceToWords") \
    .getOrCreate()

# Read sentences from the text file
lines_rdd = spark.sparkContext.textFile("sentences.txt")

# Use flatMap transformation to split each sentence into
words
words_rdd = lines_rdd.flatMap(lambda line: line.split())

# Collect the results
```

```
words = words_rdd.collect()

# Print the words
for word in words:
    print(word)

# Stop the SparkSession
spark.stop()
```

Output

```
Spark
is
a
powerful
distributed
computing
framework.
Python
is
widely
used
for
data
analysis
and
machine
learning.
Big
Data
technologies
are
revolutionizing
industries.
Machine
learning
datasets
to
extract
insights.
Apache
Hadoop
is
a
popular
framework
for
distributed
storage
and
processing.
Deep
learning
models
require
large
amounts
of
labeled
data.
Natural
```

algorithms help in predictive analytics. Data scientists analyze large	language processing enables computers to understand human language.

Q5)
Create a dataset having student details such as (name, subject, score), from this dataset group students by subject using the groupBy transformation

Ans)

student_details.txt

```
Alice, Math, 85
Bob, Science, 78
Charlie, Math, 92
Alice, Science, 90
Bob, Math, 88
Charlie, Science, 82
Eve, Math, 95
Eve, Science, 85
```

groupStudents.py

```
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("GroupByExample") \
    .getOrCreate()
```

```
# Read the dataset into a DataFrame
df = spark.read.csv("student_details.txt", header=False,
inferSchema=True) \
    .toDF("name", "subject", "score")

# Group students by subject
grouped_df = df.groupBy("subject").agg({"name":
"collect_list", "score": "avg"})

# Show the result
grouped_df.show()

# Stop the SparkSession
spark.stop()
```

Output

```
+--------+------------------+------------------+
|subject |collect_list(name)|       avg(score)|
+--------+------------------+------------------+
|Math    |[Alice, Charlie, Eve, Bob]|90.0|
|Science |[Bob, Charlie, Eve, Alice]|83.75|
+--------+------------------+------------------+
```

Q6)
From the employee dataset, collect the first 5 records as an array using the collect action.

Ans)

Code

```
from pyspark.sql import SparkSession
```

```
# Create a SparkSession
spark = SparkSession.builder \
    .appName("CollectExample") \
    .getOrCreate()

# Define the employee dataset
data = [
    (24, "John", 26, 30000),
    (34, "Jack", 40, 80000),
    (61, "Joshi", 25, 35000),
    (45, "Jash", 35, 75000),
    (34, "Yash", 40, 60000),
    (67, "Smith", 20, 24000),
    (42, "Lion", 42, 56000),
    (62, "Kate", 50, 76000),
    (21, "Cassy", 51, 40000),
    (10, "Ronald", 57, 65000),
    (24, "John", 26, 30000),
    (67, "Smith", 20, 24000),
    (45, "Jash", 35, 75000),
    (21, "Cassy", 51, 40000)
]

# Create a DataFrame from the dataset
df = spark.createDataFrame(data, ["RegNo", "EmpName",
"Age", "Salary"])

# Collect the first 5 records as an array
first_5_records = df.take(5)

# Display the result
for record in first_5_records:
    print(record)

# Stop the SparkSession
```

```
spark.stop()
```

Output

```
Row(RegNo=24, EmpName='John', Age=26, Salary=30000)
Row(RegNo=34, EmpName='Jack', Age=40, Salary=80000)
Row(RegNo=61, EmpName='Joshi', Age=25, Salary=35000)
Row(RegNo=45, EmpName='Jash', Age=35, Salary=75000)
Row(RegNo=34, EmpName='Yash', Age=40, Salary=60000)
```

Q7)
Demonstrate the creation of RDD using Parallelized collection, existing RDD by finding the sum of all elements in an RDD1(which holds array elements). Also, create an RDD from external sources

Ans)

input.txt

```
This is a sample text file.
It contains multiple lines of text.
Each line represents a sentence.
You can replace this content with your own text file content.
```

rdd.py

```
from pyspark import SparkContext, SparkConf

# Create a SparkContext
conf = SparkConf().setAppName("RDDCreationDemo")
sc = SparkContext(conf=conf)

# 1. Creating an RDD using Parallelized collection
data = [1, 2, 3, 4, 5]
rdd1 = sc.parallelize(data)

# 2. Finding the sum of all elements in RDD1
```

```
sum_of_elements = rdd1.reduce(lambda x, y: x + y)
print("Sum of elements in RDD1:", sum_of_elements)

# 3. Creating an RDD from external sources
# For example, let's create an RDD from a text file
external_rdd =
sc.textFile("file:///home/Ashrut/Week5/Q7/external_file.t
xt")

# Performing an action to trigger the execution of the
Spark job
num_lines = external_rdd.count()
print("Number of lines in the external RDD:", num_lines)

# Stop the SparkContext
sc.stop()
```

Output

```
Sum of elements in RDD1: 15

Number of lines in the external RDD: 5
```

Week7 (Spark & Scala)

Q1)
Assume you have a CSV file named clickstream_data.csv with the following columns: user_id , page_id, timestamp, action (e.g., 'click', 'view', 'purchase').

- Load the data into a PySpark DataFrame.
- Display the schema and the first 5 rows of the DataFrame.

• Calculate the total number of clicks, views, and purchases for each user.
• Identify the most common sequence of actions performed by users (e.g., click -> view -> purchase).

Ans)

clickstream_data.csv

```
user_id,page_id,timestamp,action
1,101,2024-04-01 08:00:00,click
1,102,2024-04-01 08:05:00,view
2,103,2024-04-01 08:10:00,click
2,104,2024-04-01 08:15:00,click
3,105,2024-04-01 08:20:00,purchase
1,106,2024-04-01 08:25:00,view
2,107,2024-04-01 08:30:00,view
3,108,2024-04-01 08:35:00,click
3,109,2024-04-01 08:40:00,view
1,110,2024-04-01 08:45:00,click
```

clickstreamAnalysis.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import count, col, concat,
lit, lag
from pyspark.sql.window import Window
from pyspark.sql.types import StructType, StructField,
StringType, TimestampType

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Clickstream Analysis") \
    .getOrCreate()
```

```
# Define the schema for the CSV file
schema = StructType([
    StructField("user_id", StringType(), True),
    StructField("page_id", StringType(), True),
    StructField("timestamp", TimestampType(), True),
    StructField("action", StringType(), True)
])

# Load data into DataFrame
clickstream_df =
spark.read.csv("file:///Ashrut/Weel7/Q1/clickstream_data.
csv", header=True, schema=schema)

# Display the schema and first 5 rows
clickstream_df.printSchema()
clickstream_df.show(5)

# Calculate total number of clicks, views, and purchases
for each user
action_counts_df = clickstream_df.groupBy("user_id",
"action").agg(count("*").alias("count"))
action_counts_df.show()

# Identify the most common sequence of actions performed
by users
w = Window.partitionBy("user_id").orderBy("timestamp")
clickstream_df = clickstream_df.withColumn("prev_action",
lag("action").over(w))

sequence_df = clickstream_df.groupBy("user_id", "action",
"prev_action") \
    .agg(count("*").alias("count")) \
    .withColumn("sequence", concat(col("prev_action"),
lit(" -> "), col("action"))) \
    .orderBy(col("count").desc())
```

```
most_common_sequence = sequence_df.select("user_id",
"sequence", "count").limit(1)
most_common_sequence.show()

# Stop SparkSession
spark.stop()
```

Output

```
root
 |-- user_id: string (nullable = true)
 |-- page_id: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- action: string (nullable = true)
```

```
+-------+-------+-------------------+--------+
|user_id|page_id|          timestamp|  action|
+-------+-------+-------------------+--------+
|      1|    101|2024-04-01 08:00:00|   click|
|      1|    102|2024-04-01 08:05:00|    view|
|      2|    103|2024-04-01 08:10:00|   click|
|      2|    104|2024-04-01 08:15:00|   click|
|      3|    105|2024-04-01 08:20:00|purchase|
+-------+-------+-------------------+--------+
only showing top 5 rows
```

```
+-------+--------+-----+
|user_id|  action|count|
+-------+--------+-----+
|      1|   click|    2|
|      3|    view|    1|
|      3|   click|    1|
|      1|    view|    2|
|      2|   click|    2|
|      2|    view|    1|
|      3|purchase|    1|
+-------+--------+-----+
```

```
+-------+-------------+-----+
|user_id|     sequence|count|
+-------+-------------+-----+
|      1|click -> view|    1|
+-------+-------------+-----+
```

Q2)
Consider a scenario of Web Log Analysis. Assume you have a log file named web_logs.txt with the columns: Timestamp, user_id, page_id, action (e.g., 'click', 'view', 'purchase'). Identify the most engaged users by calculating the total time spent on the website for each user. Implement the mentioned case with “PySpark Scala”

Ans)

web_logs.txt

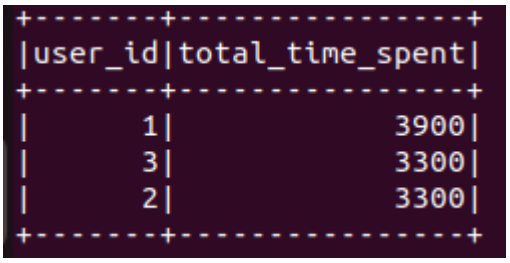
```
Timestamp,user_id,page_id,action
2024-04-01T08:00:00,101,123,click
2024-04-01T08:02:30,102,124,view
2024-04-01T08:05:45,101,125,click
2024-04-01T08:10:20,103,126,purchase
2024-04-01T08:15:10,102,127,click
2024-04-01T08:20:30,101,128,click
2024-04-01T08:25:15,103,129,view
2024-04-01T08:30:40,101,130,click
2024-04-01T08:35:50,102,131,view
2024-04-01T08:40:05,103,132,click
```

Code

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, unix_timestamp,
max, min

# Create SparkSession
spark = SparkSession.builder \
    .appName("Web Log Analysis") \
    .getOrCreate()

# Read the web logs file into a DataFrame
```

```
logs_df = spark.read \
    .option("header", "false") \
    .csv("web_logs.txt") \
    .toDF("timestamp", "user_id", "page_id", "action")

# Convert timestamp to Unix timestamp
logs_df = logs_df.withColumn("timestamp",
unix_timestamp(col("timestamp")))

# Calculate total time spent on website for each user
user_engagement = logs_df.groupBy("user_id") \
    .agg((max(col("timestamp")) -
min(col("timestamp"))).alias("total_time_spent"))

# Show the most engaged users
user_engagement.orderBy(col("total_time_spent").desc()).s
how()

# Stop SparkSession
spark.stop()
```

Output



```
+-------+----------------+
|user_id|total_time_spent|
+-------+----------------+
|      1|            3900|
|      3|            3300|
|      2|            3300|
+-------+----------------+
```

Week8 (Spark & Scala)

Q)
Consider a Spark dataframe as shown below, Need to replace a string in column Cardtype from Checking->Cash using PySpark and Spark with scala.

Hint:
Method 1: na.replace
Method 2: using regexp_replace

Ans)
Dataframe

Customer_NO	Card_type	Date	Category	Transaction Type	Amount
1000210	Platinum Card	3/17/2018	Fast Food	Debit	23.34
1000210	Silver Card	3/19/2018	Restaurants	Debit	36.48
1000210	Checking	3/19/2018	Utilities	Debit	35
1000210	Platinum Card	3/20/2018	Shopping	Debit	14.97
1000210	Silver Card	3/22/2018	Gas & Fuel	Debit	30.55
1000210	Platinum Card	3/23/2018	Credit Card Payment	Debit	559.91
1000210	Checking	3/23/2018	Credit Card Payment	Debit	559.91

Code

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField,
StringType, DoubleType
from pyspark.sql.functions import col, regexp_replace

# Initialize Spark session
spark = SparkSession.builder \
    .appName("Replace String in Spark DataFrame") \
    .getOrCreate()
```

```
# Sample data
data = [
    (1000210, "Platinum Card", "3/17/2018", "Fast Food",
"Debit", 23.34),
    (1000210, "Silver Card", "3/19/2018", "Restaurants",
"Debit", 36.48),
    (1000210, "Checking", "3/19/2018", "Utilities",
"Debit", 35.0),  # Updated to floating-point number
    (1000210, "Platinum Card", "3/20/2018", "Shopping",
"Debit", 14.97),
    (1000210, "Silver Card", "3/22/2018", "Gas & Fuel",
"Debit", 30.55),
    (1000210, "Platinum Card", "3/23/2018", "Credit Card
Payment", "Debit", 559.91),
    (1000210, "Checking", "3/23/2018", "Credit Card
Payment", "Debit", 559.91)
]

# Define schema
schema = StructType([
    StructField("Customer_NO", StringType(), True),
    StructField("Card_type", StringType(), True),
    StructField("Date", StringType(), True),
    StructField("Category", StringType(), True),
    StructField("Transaction_Type", StringType(), True),
    StructField("Amount", DoubleType(), True)
])

# Create DataFrame
df = spark.createDataFrame(data, schema=schema)

# Method 1: Using na.replace
df = df.na.replace(['Checking'], ['Cash'], 'Card_type')

# Method 2: Using regexp_replace
```

```
df = df.withColumn('Card_type',
regexp_replace(col('Card_type'), 'Checking', 'Cash'))

# Show updated DataFrame
df.show(truncate=False)

# Stop Spark session
spark.stop()
```

Output

```
+-----------+-------------+---------+-------------------+----------------+------+
|Customer_NO|Card_type    |Date     |Category           |Transaction_Type|Amount|
+-----------+-------------+---------+-------------------+----------------+------+
|1000210    |Platinum Card|3/17/2018|Fast Food          |Debit           |23.34 |
|1000210    |Silver Card  |3/19/2018|Restaurants        |Debit           |36.48 |
|1000210    |Cash         |3/19/2018|Utilities          |Debit           |35.0  |
|1000210    |Platinum Card|3/20/2018|Shopping           |Debit           |14.97 |
|1000210    |Silver Card  |3/22/2018|Gas & Fuel         |Debit           |30.55 |
|1000210    |Platinum Card|3/23/2018|Credit Card Payment|Debit           |559.91|
|1000210    |Cash         |3/23/2018|Credit Card Payment|Debit           |559.91|
+-----------+-------------+---------+-------------------+----------------+------+
```

Week9(HIVE)

Q1)
Consider the given Employee data with the attributes employee_id, birthday, first_name, family_name, gender, work_day. Perform the basic HiveQL operations as follows:

1. Create database with the name Employee.

```
CREATE DATABASE IF NOT EXISTS Employee;
```

2. Display available databases.

```
SHOW DATABASES;
```

3. Choose the Employee database and Create external and internal table into it.

```
USE Employee;

-- Create external table
CREATE EXTERNAL TABLE IF NOT EXISTS employee_external (
    employee_id INT,
    birthday DATE,
    first_name STRING,
    family_name STRING,
    gender STRING,
    work_day DATE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'Ashrut/Week9/employee_data';

-- Create internal table
CREATE TABLE IF NOT EXISTS employee_internal (
    employee_id INT,
    birthday DATE,
    first_name STRING,
    family_name STRING,
    gender STRING,
    work_day DATE
);
```

4. Load the given data to both external and managed table.

```
-- Load data into external table
LOAD DATA INPATH 'Ashrut/Week9/employee_data' OVERWRITE
INTO TABLE employee_external;
```

```
-- Load data into managed table
INSERT INTO TABLE employee_internal
SELECT * FROM employee_external;
```

5. Perform partitioning by considering gender as a partition key.

```
-- Partition external table by gender
ALTER TABLE employee_external ADD PARTITION
(gender='male') LOCATION
'/user/Ashrut/Week9/employee_external/gender=male';
ALTER TABLE employee_external ADD PARTITION
(gender='female') LOCATION
'/user/Ashrut/Week9/employee_external/gender=female';

-- Partition internal table by gender
CREATE TABLE employee_internal_partitioned (
    employee_id INT,
    birthday DATE,
    first_name STRING,
    family_name STRING,
    work_day DATE
)
PARTITIONED BY (gender STRING)
STORED AS ORC;
INSERT INTO TABLE employee_internal_partitioned PARTITION
(gender)
SELECT employee_id, birthday, first_name, family_name,
work_day, gender FROM employee_internal;
```

6. Create the buckets with suitable size.

```
-- Create buckets for internal table
CREATE TABLE employee_internal_bucketed (
    employee_id INT,
```

```
    birthday DATE,
    first_name STRING,
    family_name STRING,
    gender STRING,
    work_day DATE
)
CLUSTERED BY (employee_id) INTO 5 BUCKETS
STORED AS ORC;
INSERT INTO TABLE employee_internal_bucketed SELECT *
FROM employee_internal;
```

7. Find the oldest 10 employees from both male and female category (Note:Here you will refer to partition tables for query).

```
-- Oldest 10 employees from male category
SELECT * FROM (
    SELECT * FROM employee_external WHERE gender = 'male'
ORDER BY birthday ASC LIMIT 10
) male_oldest
UNION ALL
-- Oldest 10 employees from female category
SELECT * FROM (
    SELECT * FROM employee_external WHERE gender =
'female' ORDER BY birthday ASC LIMIT 10
) female_oldest;
```

Output

employee_id	birthday	first_name	family_name	gender	work_day
7	1975-04-05	David	Miller	male	2019-10-15
5	1982-06-30	Christopher	Brown	male	2020-01-20
3	1978-03-10	Michael	Johnson	male	2020-02-28
1	1990-05-15	John	Smith	male	2020-01-01
9	1998-07-22	Ryan	Wilson	male	2020-03-05
2	1985-08-20	Jane	Doe	female	2019-12-15
6	1992-09-12	Amanda	Jones	female	2019-11-05
8	1989-12-18	Sarah	Anderson	female	2020-02-10
4	1995-11-25	Emily	Williams	female	2020-03-10
10	1980-10-08	Elizabeth	Taylor	female	2020-01-25

8. Find the oldest 10 employee by considering Employee table and compare the time taken to perform this operation between Question 7 and Question 8.

```
-- Oldest 10 employees from Employee table (not partitioned)
SELECT * FROM (
    SELECT * FROM employee_internal ORDER BY birthday ASC
LIMIT 10
) oldest_employees;
```

Output

```
+-------------+-----------+-----------+-------------+--------+------------+
| employee_id |  birthday  | first_name | family_name | gender |  work_day  |
+-------------+-----------+-----------+-------------+--------+------------+
|      7      | 1975-04-05 |   David    |   Miller    |  male  | 2019-10-15 |
|      5      | 1982-06-30 | Christopher|   Brown     |  male  | 2020-01-20 |
|      3      | 1978-03-10 |  Michael   |   Johnson   |  male  | 2020-02-28 |
|      1      | 1990-05-15 |   John     |   Smith     |  male  | 2020-01-01 |
|     10      | 1980-10-08 | Elizabeth  |   Taylor    | female | 2020-01-25 |
|      2      | 1985-08-20 |   Jane     |   Doe       | female | 2019-12-15 |
|      8      | 1989-12-18 |   Sarah    |   Anderson  | female | 2020-02-10 |
|      6      | 1992-09-12 |   Amanda   |   Jones     | female | 2019-11-05 |
|      4      | 1995-11-25 |   Emily    |   Williams  | female | 2020-03-10 |
|      9      | 1998-07-22 |   Ryan     |   Wilson    |  male  | 2020-03-05 |
+-------------+-----------+-----------+-------------+--------+------------+
```

9. Perform drop and alter operation on internal table

```
-- Drop internal table
DROP TABLE IF EXISTS employee_internal;

-- Alter internal table (add a new column)
ALTER TABLE employee_internal_partitioned ADD COLUMN
department STRING;
```

Week10(HIVE)

1. Create hbase table as per the given data.

```
create 'employee', 'personal', 'professional'
```

Output

```
0 row(s) in 1.1340 seconds
```

2. Describe the table after inserting all rows of data into it.

```
put 'employee', '1', 'personal:name', 'Angela'
put 'employee', '1', 'personal:city', 'chicago'
put 'employee', '1', 'personal:age', '31'
put 'employee', '1', 'professional:designation',
'Architect'
put 'employee', '1', 'professional:salary', '70000'

put 'employee', '2', 'personal:name', 'dwayne'
put 'employee', '2', 'personal:city', 'bostan'
put 'employee', '2', 'personal:age', '35'
put 'employee', '2', 'professional:designation', 'Web
developer'
put 'employee', '2', 'professional:salary', '65000'

put 'employee', '3', 'personal:name', 'david'
put 'employee', '3', 'personal:city', 'seattle'
put 'employee', '3', 'personal:age', '29'
put 'employee', '3', 'professional:designation',
'Engineer'
put 'employee', '3', 'professional:salary', '55000'

put 'employee', '4', 'personal:name', 'rahul'
put 'employee', '4', 'personal:city', 'USA'
put 'employee', '4', 'personal:age', '31'
```

```
put 'employee', '4', 'professional:designation',
'architect'
put 'employee', '4', 'professional:salary', '70000'

put 'employee', '5', 'personal:name', 'jony'
put 'employee', '5', 'personal:city', 'chicago'
put 'employee', '5', 'personal:age', '29'
put 'employee', '5', 'professional:designation', 'Data
analyst'
put 'employee', '5', 'professional:salary', '80000'

put 'employee', '6', 'personal:name', 'sony'
put 'employee', '6', 'personal:city', 'bostan'
put 'employee', '6', 'personal:age', '29'
put 'employee', '6', 'professional:designation', 'Data
analyst'
put 'employee', '6', 'professional:salary', '80000'

describe 'employee'
```

Output

```
Table employee is ENABLED
employee
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
6 row(s) in 0.0480 seconds
```

3. Update the salary of an empid 3 from 55000 to 65000 and describe the table to show updates.

```
put 'employee', '3', 'professional:salary', '65000'
describe 'employee'
```

Output

```
0 row(s) in 0.0170 seconds

Table employee is ENABLED
employee
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
6 row(s) in 0.0180 seconds
```

4. Retrieve employees details whose salary is greater than or equals to 70000.

```
scan 'employee', {COLUMNS => ['personal:name',
'personal:city', 'professional:designation',
'professional:salary'], FILTER =>
"(PrefixFilter('professional:salary') AND
(QualifierFilter(>=,'binary:70000')))"}
```

Output

```
ROW                                     COLUMN+CELL
 1                                      column=personal:name,
timestamp=1680698400199, value=Angela
 1                                      column=personal:city,
timestamp=1680698400358, value=chicago
 1
column=professional:designation, timestamp=1680698400520,
value=Architect
 1
column=professional:salary, timestamp=1680698400677,
value=70000
 4                                      column=personal:name,
timestamp=1680698401569, value=rahul
 4                                      column=personal:city,
timestamp=1680698401709, value=USA
 4
column=professional:designation, timestamp=1680698401870,
value=architect
 4
column=professional:salary, timestamp=1680698402027,
value=70000
 5                                      column=personal:name,
timestamp=1680698402397, value=jony
 5                                      column=personal:city,
timestamp=1680698402555, value=chicago
 5
column=professional:designation, timestamp=1680698402715,
value=Data analyst
 5
column=professional:salary, timestamp=1680698402871,
value=80000
 6                                      column=personal:name,
timestamp=1680698403232, value=sony
 6                                      column=personal:city,
```

```
timestamp=1680698403390, value=bostan
 6
column=professional:designation, timestamp=1680698403547,
value=Data analyst
 6
column=professional:salary, timestamp=1680698403703,
value=80000
4 row(s) in 0.0330 seconds
```

5. Read the personal data of an employee whose name is David.

```
get 'employee', '3', {COLUMN => ['personal:name',
'personal:city', 'personal:age']}
```

Output

```
ROW                                COLUMN+CELL
 3                                 column=personal:name,
timestamp=1680698401141, value=david
 3                                 column=personal:city,
timestamp=1680698401285, value=seattle
3 row(s) in 0.0120 seconds
```

6. Describe the employee details whose designation is data analyst.

```
scan 'employee', {COLUMNS => ['personal:name',
'personal:city', 'personal:age',
'professional:designation', 'professional:salary'],
FILTER => "(QualifierFilter(=,'binary:Data analyst'))"}
```

Output

```
ROW                                COLUMN+CELL
 5                                 column=personal:name,
timestamp=1680698402397, value=jony
```

```
 5                                    column=personal:city,
timestamp=1680698402555, value=chicago
 5
column=professional:designation, timestamp=1680698402715,
value=Data analyst
 5
column=professional:salary, timestamp=1680698402871,
value=80000
 6                                    column=personal:name,
timestamp=1680698403232, value=sony
 6                                    column=personal:city,
timestamp=1680698403390, value=bostan
 6
column=professional:designation, timestamp=1680698403547,
value=Data analyst
 6
column=professional:salary, timestamp=1680698403703,
value=80000
2 row(s) in 0.0180 seconds
```

7. Count the number of rows and columns present in the created table.

```
count 'employee'
describe 'employee'
```

Output

```
6 row(s) in 0.0180 seconds

Table employee is ENABLED
employee
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
```

```
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
6 row(s) in 0.0160 seconds
```

8. Delete the age column from personal data

```
delete 'employee', '1', 'personal:age'
delete 'employee', '2', 'personal:age'
delete 'employee', '3', 'personal:age'
delete 'employee', '4', 'personal:age'
delete 'employee', '5', 'personal:age'
delete 'employee', '6', 'personal:age'
```

Output

```
0 row(s) in 0.0270 seconds

Table employee is ENABLED
employee
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS
=> '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
=> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
6 row(s) in 0.0150 seconds
```