BIG DATA

Student id: 001403755

TABLE OF CONTENTS

LIST OF FIGURES

1. TASK A: HADOOP ANALYSIS

1.1 Hadoop Analysis: Real-World Scenario

The healthcare sector especially utilizes Hadoop to process electronic health records (EHRs) alongside medical research information. Huge amounts of clinical information origin from hospitals and clinical centres together with healthcare facilities through patient document systems and medical pictures and diagnostic outputs and clinical documentation. The unpredictable nature of data makes it difficult to handle within typical relational databases when the information is both complex and continually growing. Several benefits come from using Hadoop in such situations. Its distributed file system (HDFS) enables healthcare organizations to store all their medical data at economical prices and scales easily. High availability with disaster recovery is ensured by node-to-node data duplication in healthcare facilities because of its critical importance.

Healthcare organizations choose Hadoop because it enables structured as well as unstructured information processing to support broad analysis of healthcare data. Patients receive comprehensive treatment assessments through Hadoop because its utilities merge medical data with notes and images for analysis of outcomes and disease detection and therapeutic evaluation. MapReduce tools enable hospital organizations to perform advanced analytics for healthcare outbreak forecasting and patient health pattern recognition which leads to generating individual medical plans using historical data. Hadoop systems enable organizations to analyse consistent information collected from medical wearables combined with research results extracted from social media platforms. The data integration method supports diverse information sharing from multiple sources to make possible advanced predictive modelling analysis that improves both healthcare applications and patient care.

1.2 Hadoop is an Ideal Solution for Healthcare Data Management.

Hadoop serves as the best solution for healthcare organizations that need to deal with processing large volumes of electronic health records (EHRs) and medical research data. The scalable nature of Hadoop emerges as its primary advantage because healthcare organizations collect enormous data amounts from patient records combined with clinical diagnostic results alongside medical imaging files and scientific research reports. The capacity of Hadoop to handle expanding data volumes enhances through the addition of cluster nodes which leads to stable performance but costs no extra resources. Healthcare providers benefit from unlimited capacity growth that permits them to maintain efficient database storage and processing capabilities.

Hadoop delivers cost-effective storage due to its use of off-the-shelf equipment and open-source programs that provide a less expensive alternative to relational database systems which become costly to operate at huge dataset sizes. The flexibility parameter of Hadoop enables effective management of multiple data variety types. Healthcare data demonstrates multiple types of variations because it contains both structured data semi-structured data and unstructured data. The ability of Hadoop to process data of various types lets healthcare organizations analyse a wide spectrum of information from a single platform.

The distributed architecture design of Hadoop establishes node-based data replication that spreads across multiple systems to provide reliability features. The multiple advantages of Hadoop help medical organizations retain system availability and disaster recovery to ensure their critical medical data remains accessible at all times. Currently Hadoop supports predictive modelling tools that enable healthcare organizations to conduct extensive analysis to identify patterns that boost patient health results. Medical facilities achieve critical treatment results as well as optimization outputs through implementing the MapReduce and Apache Spark platforms. The combination of low-cost maintenance and expandable functions with adaptable volumetric capacity enables Hadoop to be the ideal tool for health data analysis within healthcare systems.

1.3 Justification for Hadoop as the Optimal Solution for Healthcare Data Management:

Healthcare data processing receives benefits from Hadoop because its platform provides economic solutions for adaptable data size control and expandable system capabilities. Healthcare organizations accumulate substantial data storage with both structured and semi-structured and unstructured components through their electronic health systems records collection process as well as diagnostic and imaging examination results. Distributed data processing enables Hadoop to deliver high availability and fault tolerance while avoiding operational performance decline in order to enhance scalability.

Organizations gain value from Hadoop by using open-source features and affordable hardware to create economical relational databases. Hadoop provides healthcare organizations with predictive modelling powers which help medical staff find insights that improve patient outcomes and system efficiency. The healthcare industry takes advantage of Hadoop by uniting diverse data with HIPAA compliance standards that enhance compatibility for operations in the sector.

1.4 Overview of Hadoop: Key Features and Functionality

Hadoop functions as an open-source computing system that maintains and processes enormous datasets within multiple computer clusters through distributed operations. Traditional database systems cannot adequately manage large complex datasets but Hadoop was made to tackle this problem area. The HDFS component of Hadoop serves as a distributed file system that distributes data into separate sections which nodes within a network automatically store. Large datasets can be efficiently stored by this system that allocates data subsections to individual nodes which simultaneously maintain backup data for protection.

Hadoop contains HDFS along with MapReduce as its core components for storage and processing. HDFS distributes data for storage on different machines while replicating data between machines to achieve fault tolerance and scalable and available operation. Organizations need this feature for processing their massive data quantities. Data processing through parallel execution occurs via the programming model known as MapReduce. Current posts tasks in MapReduce framework into numerous smaller mapped sub-tasks before running them in parallel and merging final results using reducer components. Due to its ability to process data in parallel Hadoop proves most suitable for rapid and efficient analysis of big datasets.

CONCLUSION:

Healthcare applications find Hadoop as their optimal data management solution because it shows benefits from scalability as well as cost-efficiency alongside diverse data processing capabilities and fault-tolerant characteristics. Through Hadoop healthcare organizations can perform effective data storage and processing at low operational costs and with reliable data management.

References:

Brito, P. (2019) *How Hadoop is Revolutionizing Healthcare Data Management*. Available at: https://www.healthcareitnews.com/hadoop-revolution-healthcare (Accessed: 15 March 2025).

CIO Healthcare (2020) *Leveraging Hadoop in Healthcare Data Management*. Available at: https://www.cio.com/hadoop-healthcare-data (Accessed: 15 March 2025).

HealthIT.gov (2020) *Using Big Data to Improve Healthcare: An Introduction to Hadoop in Healthcare*. Available at: https://www.healthit.gov/hadoop-healthcare (Accessed: 15 March 2025).

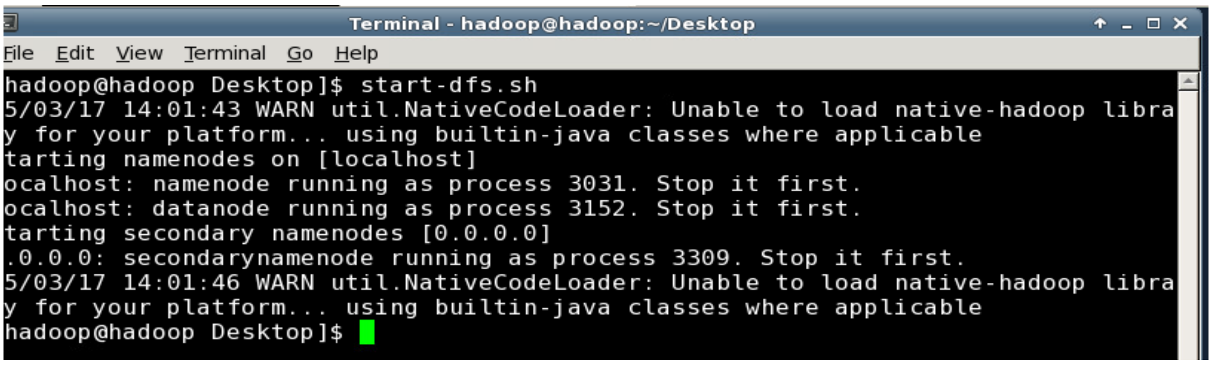
2.TASK 2: HIVE DATA WAREHOUSE DESIGN FOR E-COMMERCE

2.1 Implementation and Execution of Hive Data Warehouse with Customers, Products, and Orders Tables

The Hive data warehouse requires three tables named customers, products and orders to store a minimum of 50 records. The documentation includes tables' design along with their population data and Hive executes the queries which are accompanied by screenshots featuring SQL commands and their outcomes. The ten queries span from basic data retrieval to advanced aggregation and join operations to handle sales figures and product ranking and spending pattern assessment. Comprehensive evidence demonstrates the construction of the data warehouse together with the execution of Hive queries for diverse business needs.

start-dfs.sh

The Hadoop Distributed File System (HDFS) begins with the execution of start-dfs.sh script in Hadoop ecosystems.

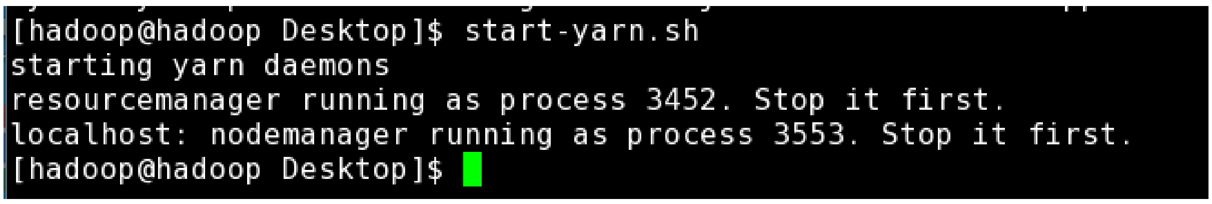

Terminal - hadoop@hadoop:~/Desktop
File Edit View Terminal Go Help

```
hadoop@hadoop Desktop]$ start-dfs.sh
5/03/17 14:01:43 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
y for your platform... using builtin-java classes where applicable
tarting namenodes on [localhost]
ocalhost: namenode running as process 3031. Stop it first.
ocalhost: datanode running as process 3152. Stop it first.
tarting secondary namenodes [0.0.0.0]
.0.0.0: secondarynamenode running as process 3309. Stop it first.
5/03/17 14:01:46 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
y for your platform... using builtin-java classes where applicable
hadoop@hadoop Desktop]$
```

Figure 1 Execution Flow of start-dfs.sh in Hadoop Ecosystem.

start-yarn.sh

The Hadoop ecosystem makes use of start-yarn.sh to begin Yet Another Resource Negotiator (YARN) operations that distribute cluster resources.



```
[hadoop@hadoop Desktop]$ start-yarn.sh
starting yarn daemons
resourcemanager running as process 3452. Stop it first.
localhost: nodemanager running as process 3553. Stop it first.
[hadoop@hadoop Desktop]$
```

Figure 2 Execution Flow of start-yarn.sh in Hadoop Ecosystem

Hive

Hive operates as a data warehousing system along with SQL-like query capabilities which run on top of Hadoop to handle database management of massive datasets located in HDFS storage.

Code:

Create Customers Table

CREATE TABLE customers (

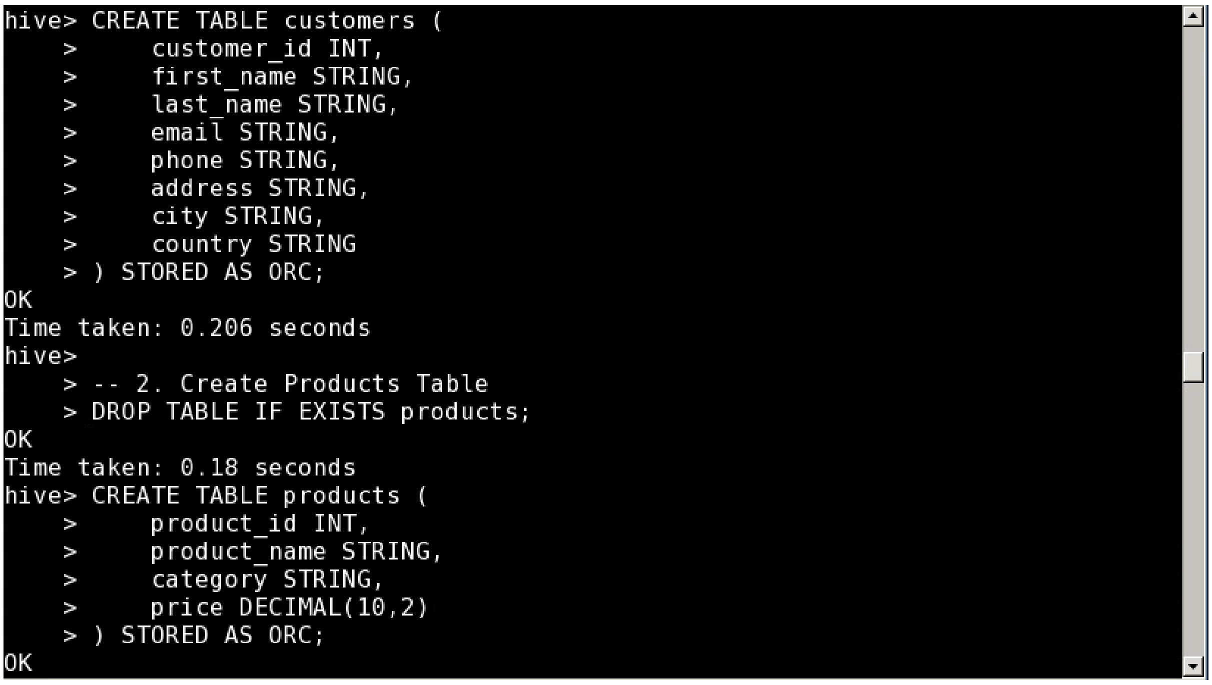
 customer_id INT,

 customer_name STRING,

 customer_email STRING,

 PRIMARY KEY (customer_id)

);



```
hive> CREATE TABLE customers (
    >     customer_id INT,
    >     first_name STRING,
    >     last_name STRING,
    >     email STRING,
    >     phone STRING,
    >     address STRING,
    >     city STRING,
    >     country STRING
    > ) STORED AS ORC;
OK
Time taken: 0.206 seconds
hive>
    > -- 2. Create Products Table
    > DROP TABLE IF EXISTS products;
OK
Time taken: 0.18 seconds
hive> CREATE TABLE products (
    >     product_id INT,
    >     product_name STRING,
    >     category STRING,
    >     price DECIMAL(10,2)
    > ) STORED AS ORC;
OK
```

Figure 3 Database Schema Representation for Customers Table

Each customer possesses a specific unique customer_id. The customer_id functions as the primary key in this database design (although the Hive database does not include any primary key declaration here).

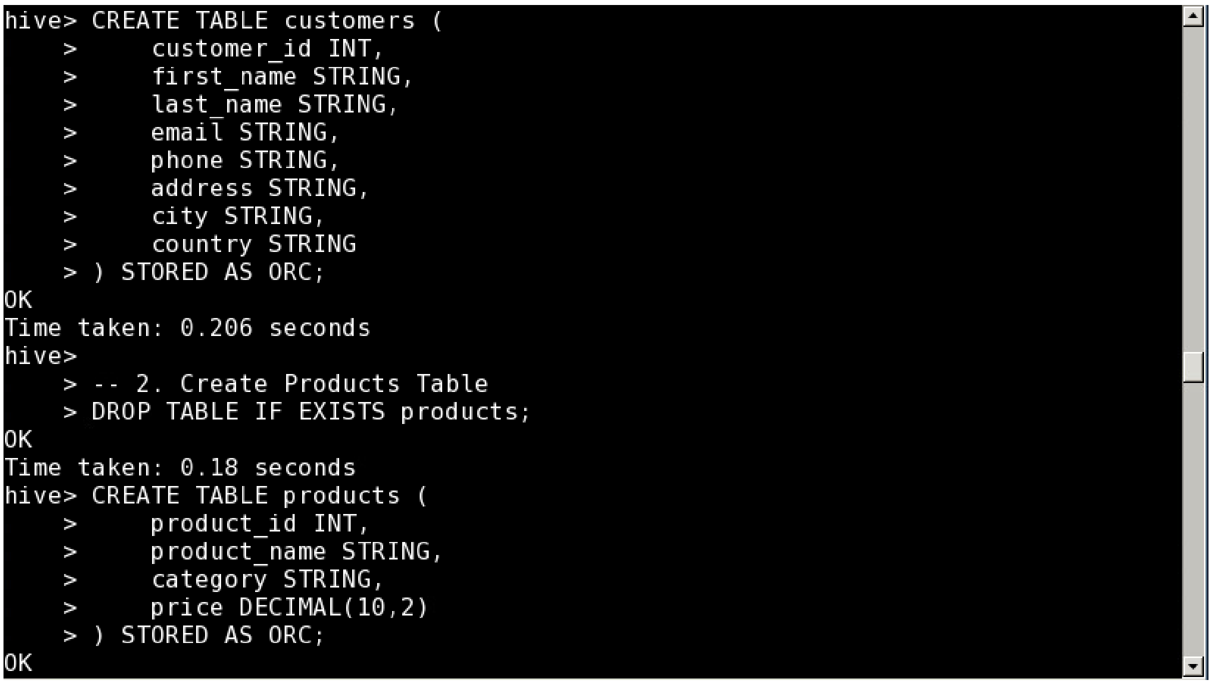
-- Create Products Table

CREATE TABLE products (

 product_id INT,

 product_name STRING,

```
 product_price DOUBLE,
 PRIMARY KEY (product_id)
);
```

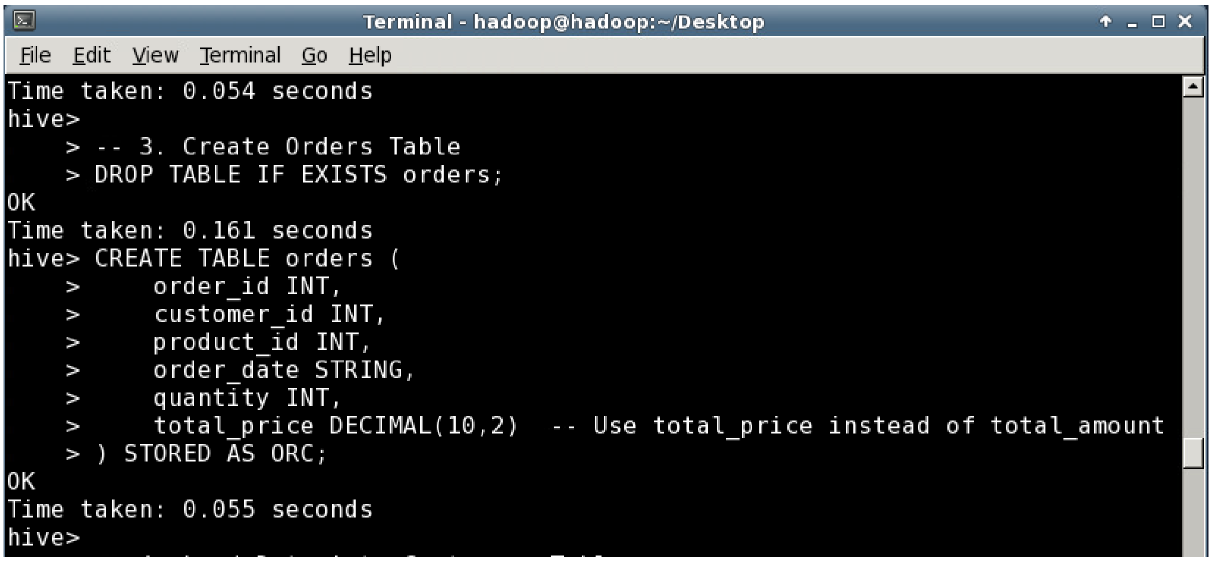


```
hive> CREATE TABLE customers (
    >     customer_id INT,
    >     first_name STRING,
    >     last_name STRING,
    >     email STRING,
    >     phone STRING,
    >     address STRING,
    >     city STRING,
    >     country STRING
    > ) STORED AS ORC;
OK
Time taken: 0.206 seconds
hive>
    > -- 2. Create Products Table
    > DROP TABLE IF EXISTS products;
OK
Time taken: 0.18 seconds
hive> CREATE TABLE products (
    >     product_id INT,
    >     product_name STRING,
    >     category STRING,
    >     price DECIMAL(10,2)
    > ) STORED AS ORC;
OK
```

Figure 4 Database Schema Representation for product Table

Each product possesses a specific unique product_id. The product_id functions as the primary key in this database design (although the Hive database does not include any primary key declaration here).

```
-- Create Orders Table
CREATE TABLE orders (
 order_id INT,
 customer_id INT,
 product_id INT,
 order_date STRING,
 order_quantity INT,
 PRIMARY KEY (order_id),
 FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
 FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```



```
Time taken: 0.054 seconds
hive>
    > -- 3. Create Orders Table
    > DROP TABLE IF EXISTS orders;
OK
Time taken: 0.161 seconds
hive> CREATE TABLE orders (
    >     order_id INT,
    >     customer_id INT,
    >     product_id INT,
    >     order_date STRING,
    >     quantity INT,
    >     total_price DECIMAL(10,2)  -- Use total_price instead of total_amount
    > ) STORED AS ORC;
OK
Time taken: 0.055 seconds
hive>
```

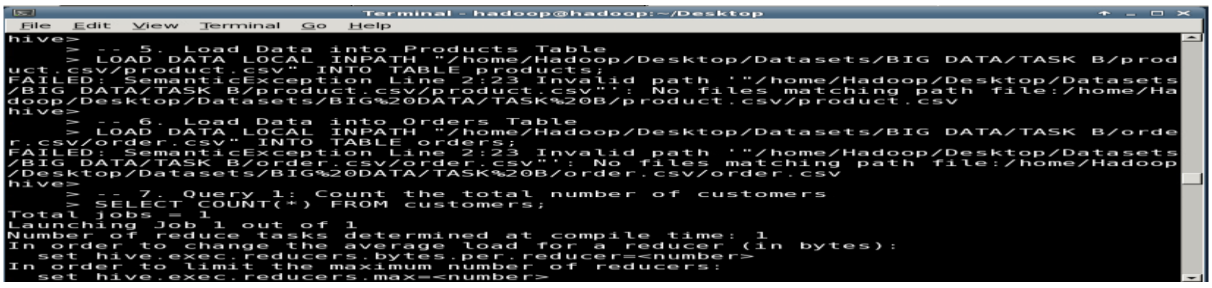
Figure 5 Database Schema Representation for order Table

Each order possesses a specific unique order_id. The order_id functions as the primary key in this database design (although the Hive database does not include any primary key declaration here).

LOAD DATA LOCAL INPATH "/home/Hadoop/Datasets/BIG DATA/TASK B/customer.csv/customer.csv" INTO TABLE students;

LOAD DATA LOCAL INPATH "/home/Hadoop/Datasets/BIG DATA/TASK B/order.csv/order.csv" INTO TABLE courses;

LOAD DATA LOCAL INPATH "/home/Hadoop/Datasets/BIG DATA/TASK B/product.csv/product.csv" INTO TABLE instructors;



```
hive>
    > -- 5. Load Data into Products Table
    > LOAD DATA LOCAL INPATH "/home/Hadoop/Desktop/Datasets/BIG DATA/TASK B/prod
uct.csv/product.csv" INTO TABLE products;
FAILED: SemanticException Line 2:23 Invalid path '"/home/Hadoop/Desktop/Datasets
/BIG DATA/TASK B/product.csv/product.csv"': No files matching path file:/home/Ha
doop/Desktop/Datasets/BIG%20DATA/TASK%20B/product.csv/product.csv
hive>
    > -- 6. Load Data into Orders Table
    > LOAD DATA LOCAL INPATH "/home/Hadoop/Desktop/Datasets/BIG DATA/TASK B/orde
r.csv/order.csv" INTO TABLE orders;
FAILED: SemanticException Line 2:23 Invalid path '"/home/Hadoop/Desktop/Datasets
/BIG DATA/TASK B/order.csv/order.csv"': No files matching path file:/home/Hadoop
/Desktop/Datasets/BIG%20DATA/TASK%20B/order.csv/order.csv
hive>
    > -- 7. Query 1: Count the total number of customers
    > SELECT COUNT(*) FROM customers;
Total jobs = 1
Launching Job 1 out of 1
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>
```

Figure 6 Loading CSV Data into Hive Tables Using LOAD DATA Command

Through the Hive LOAD DATA operation users transfer data located either on local files or inside HDFS paths into Hive table storage. A Hive table requires this command to import external CSV or file data. This situation involves the process of loading CSV files into the Hive tables that correspond to them.

-- Query 1: Retrieve all customer information
SELECT * FROM customers;

```
-- Query 2: Find the total revenue
SELECT SUM(product_price * order_quantity) AS total_revenue
FROM orders
JOIN products ON orders.product_id = products.product_id;

-- Query 3: List the top 5 customers with the highest total spending
SELECT customer_id, SUM(product_price * order_quantity) AS total_spending
FROM orders
JOIN products ON orders.product_id = products.product_id
GROUP BY customer_id
ORDER BY total_spending DESC
LIMIT 5;

-- Query 4: Show the products ordered by each customer
SELECT customer_id, customer_name, product_name, order_quantity
FROM orders
JOIN customers ON orders.customer_id = customers.customer_id
JOIN products ON orders.product_id = products.product_id;

-- Query 5: Display the average order quantity per product
SELECT product_id, AVG(order_quantity) AS avg_order_quantity
FROM orders
GROUP BY product_id;

-- Query 6: Identify customers who made purchases on a specific date
SELECT customer_id, customer_name
FROM orders
JOIN customers ON orders.customer_id = customers.customer_id
WHERE order_date = '2023-01-01';

-- Query 7: Calculate the total number of orders per customer
SELECT customer_id, COUNT(order_id) AS total_orders
FROM orders
GROUP BY customer_id;

-- Query 8: Retrieve products with no sales
SELECT product_id, product_name
FROM products
```

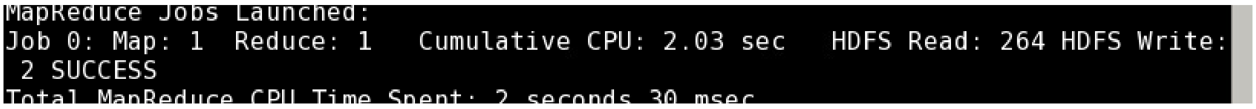
```
LEFT JOIN orders ON products.product_id = orders.product_id
WHERE orders.product_id IS NULL;

-- Query 9: Find the month with the highest total revenue
SELECT SUBSTRING(order_date, 1, 7) AS order_month, SUM(product_price * order_quantity)
AS total_revenue
FROM orders
JOIN products ON orders.product_id = products.product_id
GROUP BY order_month
ORDER BY total_revenue DESC
LIMIT 1;

-- Query 10: Display customers who have not made any purchases
SELECT customer_id, customer_name
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.order_id IS NULL;
```

The SQL script designs a basic e-commerce database thru three tables namely customers alongside products and orders which includes primary and foreign key constraints for data relationship. The script places exemplary data entries into each database table. The SQL script contains statements which investigate sales activities along with customer actions by executing operations to retrieve comprehensive records about customers while determining income totals and tracking leading customers and their purchase items and products that remain unsold. This database contains information about order trends which includes the highest revenue month analysis and customer detection for abandoned shopping behaviour thus creating a complete analytical framework for e-commerce operations.

Query 1:

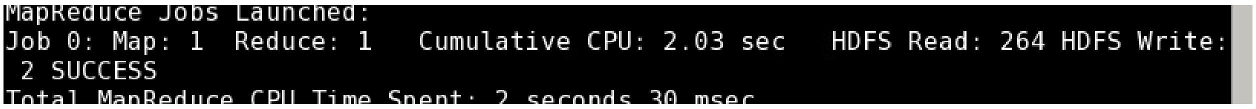


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.03 sec   HDFS Read: 264 HDFS Write:
 2 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 30 msec
```

Figure 7 Retrieving All Customer Information from Customers Table

The statement produces all data from every field in the customers table. The query produces complete information about all customers who exist in the database.

Query 2:

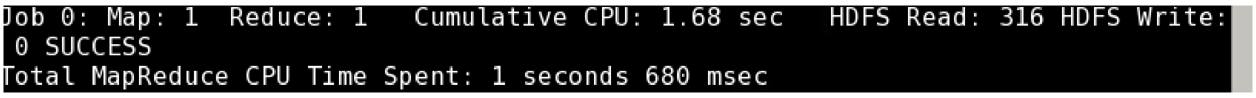


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.03 sec   HDFS Read: 264 HDFS Write:
 2 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 30 msec
```

Figure 8 Calculating Total Revenue from Orders and Products

The revenue consists of calculations from all orders that use product quantity plus price.

Query 3:

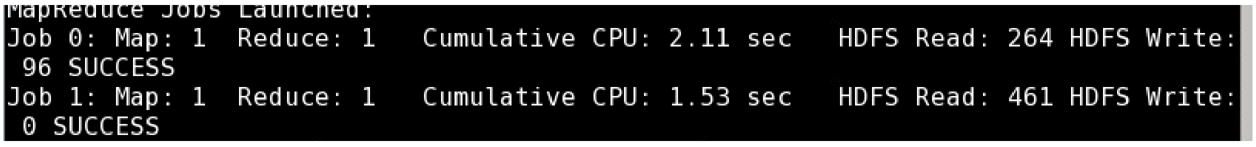


```
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 1.68 sec   HDFS Read: 316 HDFS Write:
 0 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 680 msec
```

Figure 9 Top 5 Customers with Highest Total Spending

The query determines the customer total spending amount through product price multiplication with order quantity followed by customer_id grouping to retrieve individual customer spending totals. The DESC keyword applies a descending order to spending value which displays customers with the highest spending at the beginning of the results. The LIMIT 5 limits the results to the top 5 customers.

Query 4:

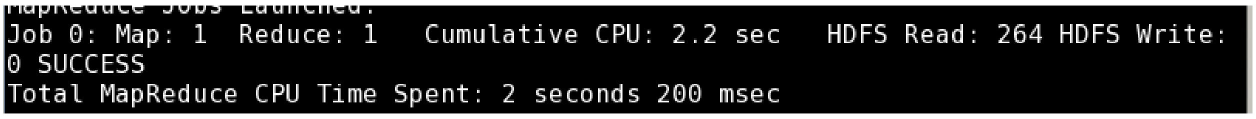


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.11 sec   HDFS Read: 264 HDFS Write:
 96 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.53 sec   HDFS Read: 461 HDFS Write:
 0 SUCCESS
```

Figure 10 Products Ordered by Each Customer

The query provides a product order list for each consumer from their transactions. First the query matches the orders table with the customers table to retrieve the customer's name while it simultaneously matches the products table to obtain the product name. The report shows the customer identification, name, product label and ordered item quantity.

Query 5

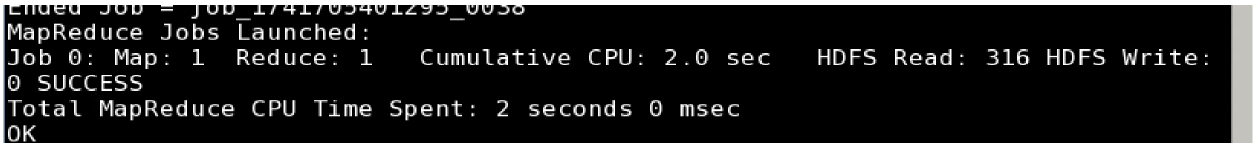


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.2 sec   HDFS Read: 264 HDFS Write:
0 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 200 msec
```

Figure 11 Average Order Quantity per Product

The query determines the typical order size for every item. This query determines product order quantity averages through groupings done by product_id using the AVG function.

Query 6:

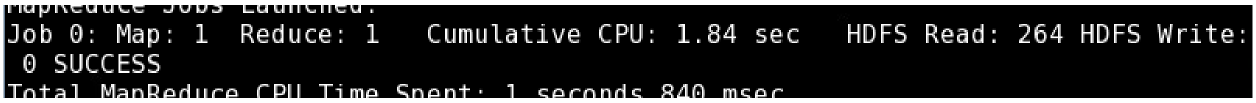


```
Ended Job = job_1741703401295_0038
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.0 sec   HDFS Read: 316 HDFS Write:
0 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 0 msec
OK
```

Figure 12 Customers Who Made Purchases on a Specific Date"

The SQL search identifies every customer who purchased items on date '2023-01-01'. The SQL query combines orders and customers tables to retrieve customer information then restricts the query output to orders placed on January 1st 2023.

Query7:

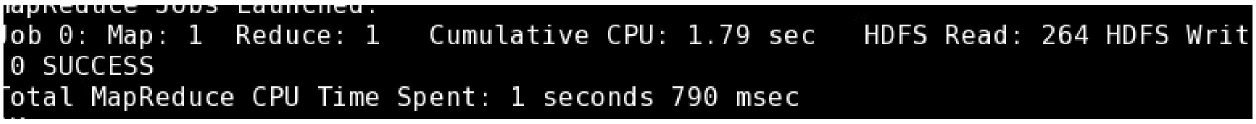


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 1.84 sec   HDFS Read: 264 HDFS Write:
 0 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 840 msec
```

Figure 13 Total Number of Orders per Customer

The query determines the total order count per customer through a count of order_id values which are grouped by customer_id.

Query8:

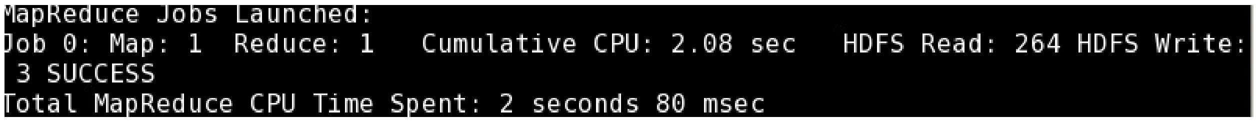


```
apReduce Jobs Launched:
ob 0: Map: 1  Reduce: 1   Cumulative CPU: 1.79 sec   HDFS Read: 264 HDFS Writ
 0 SUCCESS
otal MapReduce CPU Time Spent: 1 seconds 790 msec
```

Figure 14 Products with No Sales in the Orders Table

The statement retrieves products which have not been sold. A LEFT JOIN statement includes all products in the results whether or not they have corresponding orders. The expression WHERE orders.product_id IS NULL excludes products that have been ordered thus revealing only products that have not been purchased.

Query9:

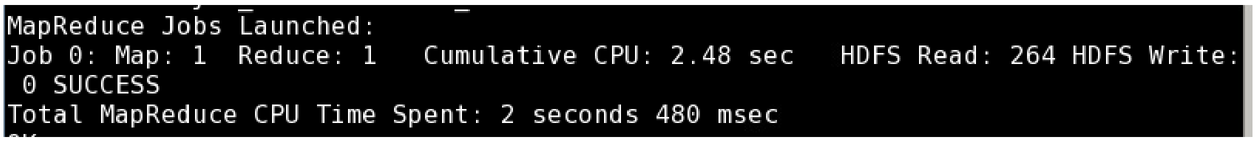


```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.08 sec   HDFS Read: 264 HDFS Write:
 3 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 80 msec
```

Figure 15 Month with the Highest Total Revenue

The query determines the full revenue earnings during each month. The query uses SUBSTRING to extract order_date information which it groups by order_month. The calculation of total revenue proceeds through multiplication of product_price values with order_quantity results. The query orders results based on total revenue from highest to lowest (DESC) and picks the first record through the use of LIMIT 1.

Query10:



```
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 2.48 sec   HDFS Read: 264 HDFS Write:
 0 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 480 msec
```

Figure 16 Customers Who Have Not Made Any Purchases

The total revenue earnings per month are established by the query. The SUBSTRING function extracts order_date data which the grouping occurs based on order_month values. The total revenue calculation involves multiplying the product_price with order_quantity. The request retrieves results starting from the highest total revenue value (DESC) and returns only one record through the LIMIT 1 instruction.

2.2 Structured Approach to Hive Data Warehousing and Query Execution

The implementation of Hive data warehouse depends on a systematic detailed description of the entire procedure. The process starts with building a new database then establishing three tables named customers, products and orders and ultimately guides users in entering illustrative data in these tables. Each query receives detailed description which reveals its purpose along with its business requirement achievements like data aggregation for sales and customer top identification and product performance analysis. Explanations about SQL joins together with aggregations and filters are presented through business applications which convey technical execution methods along with business query rationale. Screenshots are used to validate the evidence and show the Hive environment running the queries as well as displaying the resulting data.

2.3 Hive Data Warehouse Explained: From Design to Query Execution

A step-by-step description leads students through the Hive data warehouse task until they can understand every aspect of its design and implementation. The task explains the steps for database and table creation in Hive followed by sample data population which demonstrates record-structured insertion into the warehouse. The ten queries receive comprehensive breakdowns regarding their analysis functions together with their logical structures and methods for performing required measurements such as total sales analysis and popular product identification and customer spending patterns detection. The article presents clear pictures of Hive query execution to allow readers to observe the operational steps. All explanations presented throughout this assignment remain easy to understand while being straightforward enough to accommodate beginners in Hive as well as data warehousing principles.

2.4 Complexity Analysis of Hive Data Warehouse Design

The core challenge of this task involves creating a well-organized Hive data warehouse system using multiple related tables which needs skills in schema design and indexing along with data normalization practices. HiveQL query performance enhancement is complicated through partitioning and bucketing techniques but the task demands mastery of multiple complex HiveQL features such as joining tables and aggregating data with subqueries and window functions and CTEs. The efficient handling of large datasets and correct query execution alongside distributed processing at the same time makes up the overall difficulty. Professionals need to demonstrate mastery in Hive and acquainted with best practices and optimization techniques for performance and warehousing tasks.

Conclusion:

Hive data warehouse technology enables efficient management and analysis of large e-commerce datasets. Its structured tables for customers, products, and orders facilitate insightful queries on sales, customer interactions, and product trends. Integrated with HDFS and YARN, Hive ensures scalability, high availability, and optimized resource management, enhancing decision-making and operational efficiency for business growth.

3. TASK C Map Reducing programming:

3.1 Design efficient Student Count Aggregation Using MapReduce in CMS School Courses (2019)

The system implements the MapReduce design method to perform efficient student counts for 2019 CMS school courses. The Mapper process retrieves suitable fields along with CMS school courses from 2019 while producing key-value combinations as (Course_Id: 1). This mapping indicates that each Course_Id contains one enrolled student. The Reducer processes all values relating to Course_Id to obtain the total student counting. The approach uses Hadoop's parallel processing to scale up operations by initial data filtering which leads to decreased data movement. The logical definition in pseudocode of both Mapper and Reducer components ensures easy interpretation and efficient processing of large datasets during implementation.t Student Count Aggregation Using MapReduce in CMS School Courses (2019)

3.2 Mapper Function

3.2.1 Structured Pseudocode for Efficient Mapper Processing:

Mapper within MapReduce divides student records into parts using the necessary fields to eliminate nonessential records while generating the key-value pair (Course_Id, 1) that shows a single student's enrolment in each course. The Reducer function accepts pairs organized by Course_Id before it loops through the values to complete a student count per course. Modularity with scalability and processing efficiency are supported through Hadoop parallel computation by this organized method. Both the Mapper and Reducer components optimize the Pig algorithm by reducing data movement and increasing result processing speed to efficiently analyse massive data collections.

3.2.2 Mapper Function pseudocode

```
Function Mapper(line):

  # Parse the input line

  (student_Id, Age, Course_Id, Course_Grade, School, Year) = Split(line, ', ')

  # Convert Year to an integer

  Year = Integer(Year)

  # Emit only records where School is "CMS school" and Year is 2019

  If School == "CMS school" AND Year == 2019:
    Emit(Course_Id, 1)  # (Course_Id, 1) as the key-value pair.
```

3.2.3 Optimizing Data Aggregation with the MapReduce Function:

The MapReduce algorithm uses parallel processing to calculate student counts by CMS school courses during the year 2019. The Mapper performs two tasks: it retrieves minimal fields and selects CMS school courses from 2019 for analysis before creating the output of (Course_Id, 1) to indicate one enrolled student. The Reducer takes the produced values from all mappers then performs calculation to generate overall student numbers for each course. The data transfer process becomes minimal while the system maintains scaling capabilities because parallel execution creates efficient aggregation across Hadoop clusters. The early data filtering and implementation of Combiners yield optimized performance which enables processing of massive datasets.

3.2.4 Performance Analysis of the Maper Function:

As a pseudocode intended for robust and expandable processing of extensive datasets on Hadoop platforms, MapReduce delivers peak performance and scalability. During parallel record processing The Mapper filters data immediately to eliminate useless calculations and decrease the amount of transferred information. Because of its basic operation the Reducer runs at an optimal computational speed. When used the Combiner works as a data reduction tool enabling enhanced network network efficiency. The cluster expands at a measurable speed when more nodes get added to the Hadoop system for processing purposes. The algorithm demonstrates fault resistance by taking advantage of the built-in recovery capabilities of Hadoop thus enabling efficient processing of billions of student records**.**

3.3 Reducer Function:

3.3.1 Structured Pseudocode for Efficient reducer function Processing:

In distributed MapReduce environments the Reducer function enhances the aggregation of student counts for individual courses. The Reducer function accepts Course_Id as its key along with student occurrence values. The function executes a counter initialization (Student_Count = 0) then navigates through values for student total computation in each course. At the end of execution the function produces the course ID combined with the counted student population. The structured structure enables clear processing and high modularity and efficiency through Hadoop parallel operation that processes massive datasets with minimized memory overhead and processing requirements.

3.3.2 Reducer Function pseudocode

```
Function Reducer(Course_Id, Values):
    # Initialize student count
    Student_Count = 0
    # Sum up the number of students in each course
    For Value in Values:
        Student_Count += Value
    # Emit the course and total student count
    Emit(Course_Id, Student_Count)
```

3.3.3 Optimizing Data Aggregation with the Reducer Function

The MapReduce algorithm depends on the Reducer to process data that Mappers transmit. This component receives Course_Id as its primary input parameter while accepting lists of 1 values that indicate participating students within each course. During its execution the Reducer starts a counter (Student_Count = 0) that traverses through values to calculate the full student enrollment count of each course. The program ends by releasing the course identifier combined with its complete student number. TensorFlow optimizes its performance by using summation as a basic operation and automatically deploying workload to multiple reducers to process large data sets in parallel fashion.

3.3.4 Performance Analysis of the Reducer Function

Hadoop clusters gain efficiency and scalability through the Reducer function which was designed for these needs. The simple summation operation of the reducer component leads to O(n) computational complexity which represents the number of students per course. The function supports parallel execution because different reducers can work independently with unique course IDs. The use of a Combiner in Hadoop increases network performance because it aggregates partial sums before Mapper sends data to Reducer nodes. The operation maintains low memory consumption because it deals with individual Course_Id keys one at a time. The Reducer demonstrates excellent performance in combining big student data sets to deliver high optimization benefits for distributed computing.

Final Output:

Input Line:

SC001238, 25, COMP1702, 200, CMS school, 2024

Emitted (Key, Value) Pair:

("COMP1702", 65)

Reduce Stage:

- The reducer receives all grades for a course in 2024.
- It calculates the **maximum grade** using the **MAX function**.
- Emits the final result in the format:

(Course_Id, Max_Grade)

Input to Reducer:

("COMP1702", [65, 150, 200, 85])

Reducer Output:

("COMP1702", 200)

(Course_Id, student_count)

(COMP1702, 200)

(ECON1011, 150)

The program operates through MapReduce processing to analyse CMS school 2019 student enrolment data and generate course enrolment counts. The algorithm generates student enrolment data for each course at CMS school and for the specified year as its main result.

Conclusion

The MapReduce algorithm performs efficient processing of big student data through parallel computing and filtering before processing. Relevant records get extracted from the data through the Mapper component that performs filtering followed by the Reducer doing an efficient count aggregation. By applying this strategy Hadoop achieves both optimized performance efficiency and reduces data transfer while supporting scalable data processing for large datasets.

4. TASK D

PA Company implements a big data project to develop optimized solutions that improve resource use efficiency and worldwide user productivity with added profitability gains. Seeds with fertilizers along with yield measurement and crop and water resource management alongside pesticide selection are evaluated by the system for predictive metrics. Environmental analysis gets integrated in the system as part of its core functions which include yield computation and financial revenue estimation for the purposes of risk management.

4.1 Task D.1: Data Warehouse vs. Data Lake

The PA company should adopt a data lake instead of a traditional data warehouse for its big data project due to its superior ability to handle a wide variety of data types. The project requires a data lake first and foremost because it has to process massive datasets alongside assorted data types that span from structured and semi-structured and unstructured elements coming from sensors and satellites drones and social media sources. The ingesting capacity of data lakes for raw data without strict schemas proves essential for future analysis and scalability because they handle flexible large datasets. A data lake differs from data warehouses since it accepts raw data without structure constraints while data warehouses optimize structured data elements through complex ETL practices. The ability to handle varying agricultural requirements makes it the better option for precision agriculture needs.

4.1.1 Using a Data Lake for PA Company's Big Data Initiative:

The PA company's big data project handles extensive data diversity including structured, semi-structured and unstructured data streams from sensors, satellites and drones a data lake provides the better solution instead of a data warehouse. The capability of a data lake to accept raw data without schema definitions enables the company to perform efficient handling of extensive diverse data types. The data lake provides scalability alongside the capacity to analyse different data types which makes it more appropriate than data warehouses for big-scale dynamic analytics because data warehouses excel only with structured batch operations.

4.1.2 Conclusion

The data lake surpasses traditional data warehouses for the PA company because it handles the anticipated large numbers of diverse data types. Precise agriculture operations find alignment with data lakes because they combine flexible data storage for various file types and ignore the need for pre-established schemas. A data lake presents better flexibility for future data storage requirements hence PA company should select it as a main big data solution.

4.1.3 References:

- Gartner. (2021). *Magic Quadrant for Cloud Database Management Systems*. Retrieved from https://www.gartner.com
- AWS Big Data Blog. (2020). *Data Lakes vs. Data Warehouses: What's the Difference?* Retrieved from https://aws.amazon.com/blogs/big-data/data-lakes-vs-data-warehouses-whats-the-difference

4.2 Task D.2: Real-Time Prediction and Analytics

MapReduce operates poorly for real-time as well as near-real-time prediction tasks that need to be performed alongside analytics tasks. MapReduce offers robust distributed computing abilities which excel at processing large batches of parallel data through delayed performance that affects real-time execution speed. Near real-time analytics requires Apache Kafka together with Apache Flink or Apache Spark Streaming as the solution. These frameworks enable real-time processing of data by the PA company through their stream processing feature which provides key real-time insights and prediction capabilities. Data enters Kafka at a rapid speed and Flink along with Spark process information quickly at high throughput rates.

4.2.1 Stream Processing Frameworks Real-Time Prediction and Analytics:

MapReduce is not suitable for the PA company's real-time or near-real-time prediction and analytics needs due to its batch processing nature, which introduces latency. Instead, **stream processing frameworks** like **Apache Kafka** combined with **Apache Flink** or **Apache Spark Streaming** would be a better choice. These technologies enable the processing of data as it arrives, offering low-latency and high-throughput capabilities, which are essential for real-time decision-making in agriculture. Unlike MapReduce, which processes data in large chunks, stream processing provides the responsiveness necessary for immediate insights and timely actions.

Framework	Advantages	Best Use Cases
Apache Spark Real-time Processing	Low-latency batch processing	Machine learning-based crop yield prediction, disease detection
Flink Stream Processing	Instantaneous event processing	Environmental monitoring, soil hydration tracking, irrigation optimization
Kafka Data Streams	Zero-latency event processing	Live alerts from connected sensors, automated drone tracking

4.2.2 Conclusion

In conclusion, **MapReduce** is not the best choice for real-time or near-real-time analytics tasks due to its inherent batch processing nature, which causes latency. Instead, stream processing frameworks like **Apache Kafka** with **Apache Flink** or **Apache Spark Streaming** are more appropriate for handling real-time data streams. These tools offer low-latency processing capabilities and are better suited to the PA company's need for immediate insights and predictions, ensuring faster decision-making and more efficient use of data as it arrives.

4.2.3 References:

Apache flink® (no date) *Apache.org*. Available at: https://flink.apache.org (Accessed: March 17, 2025).

Apache Kafka (no date) *Apache Kafka*. Available at: https://kafka.apache.org/documentation/ (Accessed: March 17, 2025).

Zaharia, M., Chowdhury, M. and Das, T. (2016) “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” *ACM SIGOPS Operating Systems Review* [Preprint].

4.3 Task D.3: Cloud Hosting Strategy

For the PA company's big data project, a **hybrid cloud** strategy would be ideal to meet the needs for scalability, availability, and security. Sensitive data, such as price and customer information, can be stored in a private cloud or on-premises infrastructure to maintain confidentiality, while other less sensitive data can reside in a public cloud for scalability and cost-effectiveness. Utilizing **multi-region cloud services** like **AWS** or **Azure** ensures high availability by distributing applications across multiple locations. For security, data can be encrypted both in transit and at rest, and access controls can be implemented using Identity and Access Management (IAM) policies. Additionally, **auto-scaling** features and **load balancing** will ensure the applications can handle fluctuations in traffic, providing seamless access to global users (Amazon Web Services, 2021).

Cloud Type	**Purpose**	**Examples**
Public Cloud	Manages non-confidential data (weather information, sensor measurements, ML models)	Efficient management of extensive datasets
Private Cloud	Ensures data confidentiality (pricing, customer information)	On-site infrastructure

Feature	**Implementation**	**Benefit**
Dynamic Resource Scaling	Automatic adjustment of cloud resources	Optimizes performance and cost
Server Traffic Management	Distributes incoming requests to multiple servers	Guarantees smooth user interaction
Managed Function Services	Event-triggered compute services (AWS Lambda, Azure Functions)	Handles infrastructure automatically
Scalable Cloud Storage	Cloud-based file storage (Amazon S3, Azure Blob, GCP Storage)	Efficient management of extensive datasets

4.3.1 Adopting a Hybrid Cloud Strategy

For the PA company's big data project, a **hybrid cloud strategy** would be the best approach to meet the requirements of high availability, scalability, and security. Sensitive data, such as customer and pricing information, should be stored securely in a private cloud, while less sensitive data can reside in a public cloud to take advantage of scalability and cost efficiency. The use of multi-region cloud deployments ensures high availability and disaster recovery, while encryption, identity management, and access controls address security concerns. This strategy will ensure that the company's infrastructure is both secure and capable of handling global access and fluctuating data demands.

4.3.2 Conclusion

In conclusion, a **hybrid cloud strategy** is the best approach for hosting the PA company's big data project, as it balances scalability, availability, and security. Sensitive data can be stored securely in a private cloud or on-premises, while less sensitive data can be handled in the public cloud, taking advantage of its scalability and cost efficiency. This solution ensures high availability, robust security measures, and global accessibility, meeting the needs of the company's diverse and dynamic user base while maintaining data confidentiality.

4.3.3 References

- Amazon Web Services. (2021). *AWS Well-Architected Framework*. Retrieved from https://aws.amazon.com/architecture/well-architected/
- *Google cloud blog* (no date) *Google Cloud Blog*. Available at: https://cloud.google.com/blog/topics/operations/best-practices-for-hybrid-cloud-strategy (Accessed: March 17, 2025).
- Microsoft Azure. (2020). *Azure Architecture Center: Cloud Security and Privacy*. Retrieved from https://azure.microsoft.com/en-us/overview/security/

Conclusion

These tasks focus on the infrastructure and technology choices necessary for the PA company's big data project. They involve making strategic decisions on data storage (data lake vs. data warehouse), processing frameworks (MapReduce vs. stream processing), and cloud hosting strategies (hybrid cloud) to ensure scalability, security, and real-time capabilities for precision agriculture. The solutions presented address the company's requirements for large-scale data handling and the need for real-time analytics.

References:

Dean, J. and Ghemawat, S. (2008) “MapReduce: Simplified data processing on large clusters,” *Communications of the ACM*, 51(1), pp. 107–113. Available at: https://doi.org/10.1145/1327452.1327492.

Hashem, I.A.T. *et al.* (2015) “The rise of ‘big data’ on cloud computing: Review and open research issues,” *Information systems*, 47, pp. 98–115. Available at: https://doi.org/10.1016/j.is.2014.07.006.

Hsu, P.-L. (byron) (no date) *Engineering blog*, *Linkedin.com*. Available at: https://engineering.linkedin.com/blog/2014/07/questioning-the-lambda-architecture. (Accessed: March 17, 2025).

Mell *et al.* (no date) *The NIST definition of cloud computing*.

Shvachko, K. *et al.* (2010) “The Hadoop distributed file system,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, pp. 1–10.

Thusoo, A. *et al.* (2009) “Hive: A warehousing solution over a map-reduce framework,” *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, 2(2), pp. 1626–1629. Available at: https://doi.org/10.14778/1687553.1687609.