Project: Traffic Data Storage and Analysis using Java and HDFS

This project involves developing a **Java-based system** to store and analyze **traffic data** using **HDFS** (Hadoop Distributed File System). It enables **efficient data retrieval and processing** without using **MapReduce or Spark**.

1️⃣ Project Overview

🚦 **Objective**:

- Store **real-time traffic data** (like vehicle count, speed, congestion levels) in **HDFS**.
- Implement **basic analytics** in Java to extract useful traffic insights (e.g., peak traffic hours, average vehicle speed).

📌 **Key Features**:
✅ **Java-based Traffic Data Processing**
✅ **Data Storage & Retrieval using HDFS**
✅ **Simple Data Analysis without MapReduce/Spark**
✅ **Efficient Log Handling**

2️⃣ Technologies Used

Component	Technology
Programming Language	Java (OOP concepts)
Big Data Storage	HDFS (Hadoop Distributed File System)
Data Handling	CSV Format
Data Processing	Java File Handling, Streams

3️⃣ System Architecture

🚗 **Traffic Sensors/Logs** → 📂 **Traffic Data Files (CSV format)** → 📥 **HDFS Storage** → 🛠️ **Java-based Processing** → 📊 **Data Analysis Output**

1️⃣ **Traffic sensors generate logs** → Stored as **CSV files**.
2️⃣ **Java application uploads these files to HDFS**.
3️⃣ **Java-based analysis extracts insights (e.g., busiest hours, avg. speed)**.

4 Setting Up the Environment

Before coding, set up HDFS and Java:

- **Hadoop Installation** (HDFS setup)

```
bash
CopyEdit
start-dfs.sh   # Start Hadoop HDFS
hdfs dfs -mkdir /traffic_data   # Create directory in HDFS
```

- **Java Setup** (JDK installed)

---

5 Project Implementation

📌 Step 1: Sample Traffic Data (traffic_data.csv)

Create a sample CSV file with the following structure:

```
timestamp,vehicle_count,avg_speed,congestion_level
2025-02-24 08:00,120,45,Moderate
2025-02-24 09:00,200,30,High
2025-02-24 10:00,80,50,Low
2025-02-24 11:00,150,40,Moderate
```

📌 Step 2: Java Code for Uploading Data to HDFS

```
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

public class UploadToHDFS {
    public static void main(String[] args) {
        String localFilePath = "traffic_data.csv";
        String hdfsFilePath = "/traffic_data/traffic_data.csv";

        Configuration conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://localhost:9000");

        try {
            FileSystem fs = FileSystem.get(conf);
            Path srcPath = new Path(localFilePath);
            Path destPath = new Path(hdfsFilePath);

            fs.copyFromLocalFile(srcPath, destPath);
            System.out.println("File uploaded to HDFS successfully!");
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

🚀 Execution:

```
javac -cp `hadoop classpath` UploadToHDFS.java
java -cp `hadoop classpath`:. UploadToHDFS
```

This will **upload** the `traffic_data.csv` file to HDFS.

📌 Step 3: Java Code for Retrieving Data from HDFS

```
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

public class ReadFromHDFS {
    public static void main(String[] args) {
        String hdfsFilePath = "/traffic_data/traffic_data.csv";

        Configuration conf = new Configuration();
        conf.set("fs.defaultFS", "hdfs://localhost:9000");

        try {
            FileSystem fs = FileSystem.get(conf);
            Path filePath = new Path(hdfsFilePath);
            BufferedReader br = new BufferedReader(new
InputStreamReader(fs.open(filePath)));

            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }

            br.close();
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

🚀 Execution:

```
javac -cp `hadoop classpath` ReadFromHDFS.java
java -cp `hadoop classpath`:. ReadFromHDFS
```

This will **read** the data stored in HDFS.

📌 Step 4: Java Code for Basic Data Analysis (Peak Traffic Hours)

```
import java.io.*;
import java.util.*;

public class TrafficAnalysis {
    public static void main(String[] args) {
        String filePath = "traffic_data.csv";
        int maxVehicles = 0;
        String peakHour = "";
```

```
        try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
            String line;
            br.readLine(); // Skip header

            while ((line = br.readLine()) != null) {
                String[] data = line.split(",");
                String time = data[0].split(" ")[1]; // Extract hour
                int vehicleCount = Integer.parseInt(data[1]);

                if (vehicleCount > maxVehicles) {
                    maxVehicles = vehicleCount;
                    peakHour = time;
                }
            }
            System.out.println("Peak Traffic Hour: " + peakHour + " with "
+ maxVehicles + " vehicles.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

🚀 Execution:

```
javac TrafficAnalysis.java
java TrafficAnalysis
```

💡 **Output Example:**

```
Peak Traffic Hour: 09:00 with 200 vehicles.
```

---

6 Testing and Execution

✅ **Step 1:** Upload CSV to HDFS
✅ **Step 2:** Retrieve Data from HDFS
✅ **Step 3:** Analyze Peak Traffic Hours

---

7 Future Enhancements

- **Use a Web Dashboard (HTML/CSS/JS) to visualize data** 📊
- **Integrate IoT sensors for real-time traffic updates** 🚗
- **Store data in a relational database (MySQL/PostgreSQL) for advanced queries**

📌 Conclusion

This project **stores and analyzes traffic data in HDFS using Java**, without **MapReduce/Spark**. It **uploads, retrieves, and processes** data using **simple Java file handling & HDFS commands**.