Pig Scripts

Pig – 1 (Single Record Lookup) – pig_single_record_lookup.pig

```
data = LOAD '$input' USING PigStorage(',') AS
(VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fw
d_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improveme
nt_surcharge,total_amount);

filtered = FILTER data BY VendorID == 2 AND tpep_pickup_datetime == '2017-10-01 00:15:30' AND
tpep_dropoff_datetime == '2017-10-01 00:25:11' AND passenger_count == 1 AND trip_distance == 2.17;

STORE filtered into '$output';
```

Pig – 2 (Filter) – pig_filter.pig

```
data = LOAD '$input' USING PigStorage(',') AS
(VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fw
d_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improveme
nt_surcharge,total_amount);

filtered = FILTER data BY RatecodeID == 4;

STORE filtered into '$output';
```

Pig – 3 (Group by Accompanied with Order by) – pig_group_with_order.pig

```
data = LOAD '$input' USING PigStorage(',') AS
(VendorID,tpep_pickup_datetime,tpep_dropoff_datetime,passenger_count,trip_distance,RatecodeID,store_and_fw
d_flag,PULocationID,DOLocationID,payment_type,fare_amount,extra,mta_tax,tip_amount,tolls_amount,improveme
nt_surcharge,total_amount);

generated = FOREACH data GENERATE payment_type;

filtered = FILTER generated BY payment_type != 'payment_type' AND payment_type != '';
```

```
grouped = GROUP filtered BY payment_type;

counted = FOREACH grouped GENERATE group, COUNT(filtered) as CNT_DATA;

ordered = ORDER counted BY CNT_DATA;

STORE ordered into '$output';
```

Spark Java Programs

Spark – 1 (Single Record Lookup) – SparkSingleRecordLookup.java

```
package com.upgrad.sparkassignment1;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class SparkSingleRecordLookup {

	public static void main(String[] args) {

		SparkConf conf = new SparkConf().setAppName("SingleRecordLookup");
		JavaSparkContext sc = new JavaSparkContext(conf);

		JavaRDD<String> fileRDD = sc.textFile(args[0]);

		JavaRDD<String> outRDD = fileRDD.filter(x -> {
			String[] str = x.split(",");
			if (str.length > 5 && str[0].equals("2") && str[1].equals("2017-10-01 00:15:30")
					&& str[2].equals("2017-10-01 00:25:11") && str[3].equals("1") &&
str[4].equals("2.17"))
```

```
                    return true;
                else
                    return false;
            });

            outRDD.saveAsTextFile(args[1]);

            sc.close();

        }

}
```

Spark – 2 (Filter) – SparkFilter.java

```
package com.upgrad.sparkassignment2;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;

public class SparkFilter {

        public static void main(String[] args) {

            SparkConf conf = new SparkConf().setAppName("FilterBy");
            JavaSparkContext sc = new JavaSparkContext(conf);

            JavaRDD<String> fileRDD = sc.textFile(args[0]);

            JavaRDD<String> outRDD = fileRDD.filter(x -> {
                String[] str = x.split(",");
                if (str.length > 6 && str[5].equals("4"))
```

```
                    return true;
                else
                    return false;
            });

            outRDD.saveAsTextFile(args[1]);

            sc.close();

    }

}
```

Spark – 3 (Group by Accompanied with Order by) – SparkGroupByOrderBy.java

```
package com.upgrad.sparkassignment3;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.JavaPairRDD;
import scala.Tuple2;

public class SparkGroupByOrderBy {
    public static void main(String[] args) {

        SparkConf conf = new SparkConf().setAppName("GroupByOrderBy");
        JavaSparkContext sc = new JavaSparkContext(conf);

        JavaRDD<String> fileRDD = sc.textFile(args[0]);

        JavaRDD<String> outRDD = fileRDD.filter(x -> {
            String[] str = x.split(",");
            if (str.length > 9 && !str[9].equals("payment_type") && !str[9].equals(""))
```

```
                        return true;
                else
                        return false;
        });

        JavaPairRDD<String, Integer> out1RDD = outRDD.mapToPair(x -> {
                String[] str = x.split(",");
                return new Tuple2<String, Integer>(str[9], 1);
        });

        JavaPairRDD<String, Integer> out2RDD = out1RDD.reduceByKey((a, b) -> a + b);
        JavaPairRDD<Integer, String> out3RDD = out2RDD.mapToPair(x -> x.swap()).sortByKey();
        JavaPairRDD<String, Integer> finalRDD = out3RDD.mapToPair(x -> x.swap());

        finalRDD.coalesce(1).saveAsTextFile(args[1]);
        sc.close();

    }

}
```

Explanation of Pig scripts

1. Pig 1 (pig_single_record_lookup.pig)

(a) First load input file with schema without datatypes
(b) Filter data based in requirements (VendorID == 2 AND tpep_pickup_datetime == '2017-10-01 00:15:30' AND tpep_dropoff_datetime == '2017-10-01 00:25:11' AND passenger_count == 1 AND trip_distance == 2.17)
(c) Pig will automatically type cast required fields being used for filtering.
(d) Store filtered output in a file

2. Pig 2 (pig_filter.pig)

(a) First load input file with schema without datatypes
(b) Filter data based in requirements (RatecodeID == 4)
(c) Pig will automatically type cast required field being used for filtering.

(d) Store filtered output in a file

3. Pig 3 (pig_group_with_order.pig)

(a) First load input file with schema without datatypes
(b) Extract payment_type from data
(c) Filter payment_type to ignore if header row (contains ‘payment_type’) and if empty (‘’)
(d) Group by payment_type
(e) Generate group and count for grouped payment_type
(f) Order by count
(g) Store output in a file

Explanation of Spark Java Programs

1. Spark 1 (SparkSingleRecordLookup.java)

(a) Setup Java Spark Context and load file into Java RDD using textFile method.
(b) Using filter operation with predicate lambda expression, filter out data based on requirement (VendorID == 2 AND tpep_pickup_datetime == '2017-10-01 00:15:30' AND tpep_dropoff_datetime == '2017-10-01 00:25:11' AND passenger_count == 1 AND trip_distance == 2.17)
(c) Save resultant RDD in a file

2. Spark 2 (SparkFilter.java)

(a) Setup Java Spark Context and load file into Java RDD using textFile method.
(b) Using filter operation with predicate lambda expression, filter out data based on requirement (RatecodeID == 4)
(c) Save resultant RDD in a file

3. Spark 3 (SparkGroupByOrderBy.java)

(a) Setup Java Spark Context and load file into Java RDD.
(b) Using filter operation with predicate lambda expression, filter out data where payment_type is not a header row (contains ‘payment_type’) and not empty (‘’).
(c) Convert Java RDD to JavaPairRDD using mapToPair transformation having (key, value) pair for each input element where key is payment_type and value is 1.
(d) Do aggregation using reduceByKey operation on JavaPairRDD and aggregate values having same keys.
(e) Swap JavaPairRDD so key becomes value and value becomes key and then perform sorkByKey operation which will sort by key in ascending order.
(f) Swap again JavaPairRDD so key becomes value and value becomes key. At this moment, keys are payment types and values are aggregated total in ascending order.
(g) Since data will be in different partitions so perform coalesce partition operation to gather data in one single partition and then save data in a file.