**ICS 490: Big Data Storage**

**Homework Assignment #5**

**Due: See Syllabus**

**Total: 50 Points**

**Objectives**

In this assignment, you will practice the following concepts:

* Map-reduce join
* Custom WritableComparable classes
* Oozie workflows

**Problem Description**

You are given two data sets that represent Customers and Orders. The schemas for the two data sets are as follows:

Customers (cusID, lastName, firstName, street, city, state, zipCode)

Orders (orderID, cusID, productId, date, Time)

You are required to implement MapReduce jobs to answer the following question:

“**Find how many times each product is sold in each zip code**”.

If you are to use a relational database management system, the following SQL query will produce the required output

select zipCode, productID, count(\*)

from Customer natural join Order

group by zipCode, productID

The following figure show that query execution plan for the previous query consists mainly of two operators: join and group by. The figure also shows how the query answer can be computed using a sequence of two map reduce jobs.

**MapReduce Job 2:**

count(\*)G(zipCode,productID)

Customers

{cusId,zipCode,productId}

**MapReduce Job 1:**

∏cusID,productID

∏cusID,zipCode

Orders

**Description of Map Reduce Jobs**

**MapReduce Job 1:**

This job will mainly execute the project and join operators:

**Mapper\_1 input:** The mapper can have two types of inputs as follows

* Lines from Customers file:

(lineNUm,(cusID, lastName, firstName, street, city, state, zipCode))

* Lines from Orders file:

(lineNUm,(orderId, cusID, productId, date, Time))

**Mapper\_1 output:**

For each tuple from Customer file, the output is: (cusID,{1,zipcode})

For each tuple from Orders file, the output is: (cusID,{2,productID})

**Reducer\_1 Input:**

**Key:** cusID

**Input Iterator:** one element of the format (1,zipcode) and zero or more elements of the format (2,productID) records based on how many products are purchased by this customer)

**Reducer\_1 Function:**

For each tuple of the format (2,productID) in the input iterator, produce one output tuple in the format (zipcode,productID).

**Reducer\_1 output:**

For each input tuple, the reducer produces 1 or more tuple of the format (zipCode,productID)

**MapReduce Job 2:**

**Mapper\_2 Input:** (line\_num, (zipCode,productID))

**Mapper\_2 output:** ((zipCode,productID),1) (the mapper is a simple mapper that reads the two numbers (integers) in the input line and use them to construct a composite output key).

**Reducer\_2 Input:** ((zipCode,productID),{1,1,1,1..})

**Reducer\_2 output:** ((zipCode,productID) sum of input ones)

**Step 1: Implement WritableComparable class:**

**Steps to complete this project**

You need to implement a WritableComparable class that can hold two integers. This is similar to the CompositeKey class that is posted on D2L as part of the WritableComparable example. We will use this class as the data type for:

* Mapper\_1 output value (value is two integers)
* Reducer\_1 input value
* Mapper\_2 output key (key is two integers)
* Reducer\_2 input key
* Reducer\_2 output key

**Step 2: Implement Map Reduce Job 1:**

The details of this job are explained above. Implement and test this job and make sure it produces the required output. Note that you will the output produced by this job in order to test the second map reduce job. The implementation of this job should be done in three files:

* Job1Mapper.java
* Job1Reducer.java
* Job1Driver.java

**Step 3: Implement Map Reduce Job 2:**

The details of this job are explained above. Implement and test this job. To test this job, use the output that was produced by job 1 as input for job 2. Make sure that job 2 produces the expected query output. The implementation of this job should be done in three files:

* Job2Mapper.java
* Job2Reducer.java
* Job2Driver.java

**Step 4: Oozie workflow:**

Start from the job.properties and workflow.xml files that are posted on D2L. Change these files as necessary to run the two jobs that you implemented above.

**What to submit**

Submit files that is your Oozie Job directory. The structure of your directory must be as follows:

* Assignment5Submssion
  + src
    - Job1Mapper.java
    - Job1Reducer.java
    - Job1Driver.java
    - Job2Mapper.java
    - Job2Reducer.java
    - Job2Driver.java
    - CustomWritable.java
  + job:
    - lib:
      * assignment5jobs.jar
    - job.properties
    - workflow.xml