Hackathon 2018 - Hierarchical Data Partitioning

Context

Solving retail problems like forecasting demand requires processing and analysis of large amounts of data (BIG Data), particularly sales data over time. One approach to managing such volumes of data is to partition it using an “adequate” partitioning strategy.

In this hackathon, you will learn about retail in general. Specifically, you will learn about retail data and how it’s structured. We will give you a set of data files and you will find a creative way to partition them into smaller chunks. Finally, you will try to come up with an “adequate” partitioning strategy on your own based on your analysis of the data. Don’t worry, you will be guided throughout the hackathon with levels that will help you incrementally build and refactor your solution ;-)!

Problem (Big Data =? Big Problems)

Retail Data

Hierarchy Data

The hierarchies are csv files in which the columns, or a subset of the columns are linked in a tree like structure. This model mandates that each child object has only one parent, whereas each parent can have one or more child objects. Typically retailers organize their product, location and calendar data in a many to one hierarchical manner. For example, consider the store to city, or day to year relationships. Examples :

```
products.csv
header = item, category, department

locations.csv
header = store, city, state, county

calendar.csv
header = day(friday_end),month, quarter, year
```

Fact Data

Fact data are measures that are a function of one or many levels of the hierarchies. Example the how many **units** were sold of a given **item** in a given **store** on a given **day**. In the example below, we store the sales in a csv file where sales units are a function of (item, store,day).

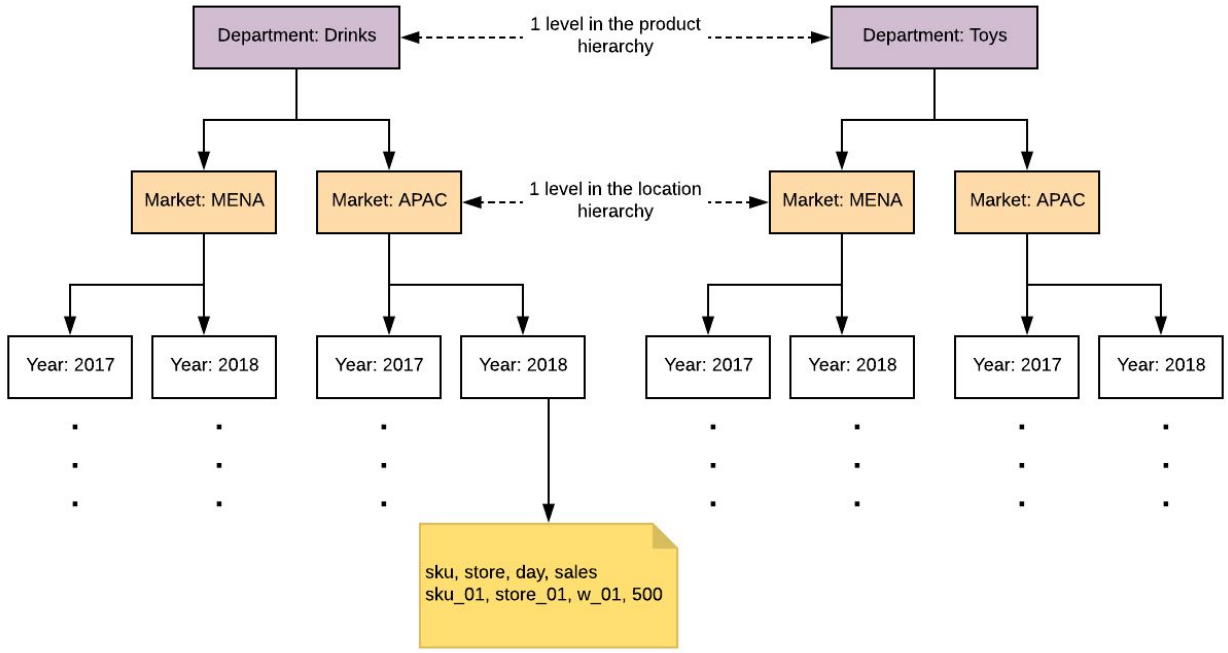
```
sales.csv
header = item, store, day(friday_end), sales
```

Goal

Textually:
Partition the sales.csv file into a hierarchy (i.e.: tree structure) of folders and files where:

- The nodes of the tree are values of only one level from all the levels in any of the Hierarchy Data files.
- The leaves of the tree are files named sales.csv and contain only sales data for the intersections of (item, store, day) that map to the containing folder subtree.

Graphically:



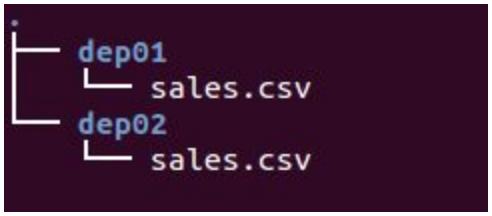
Levels

Level 1

Based on the 3 hierarchy files, using a product level name as an input (e.g.: class, department), partition the sales file into folders such that:

- Each folder is named with the id of the product level passed to the program as an input.
- In each folder, a file named sales.csv is created containing only rows that map to the level whose id is the containing folder name.

Example: using Departement level as input, the tree structure should look like the following:



Note: the output sales file should be in the same format as the input file, i.e: Don't wrap values in quotes, use ',' as field separator.

Level 2

Assuming you have the same input in level 1, if on top of that, the user supplies a parameter or an additional configuration value for a location level (e.g.: city, state, country), then on top of what you did in level 1, further partition the sales.csv file into subfolders such that:

- Each subfolder is named with the id of the location level passed to the program.
- Each subfolder contains a file named sales.csv which contains sales data for all the items that map to the grandparent folder and all the stores that map to the level whose id is the containing subfolder name.

Example : using Departement level from Product hierarchy and State level from Location hierarchy as input, the result should be in this shape:

Level 3

Same idea for level 1 and 2, but this time we introduce a new level (calendar) - see graphical representation of the goal.

Level 4

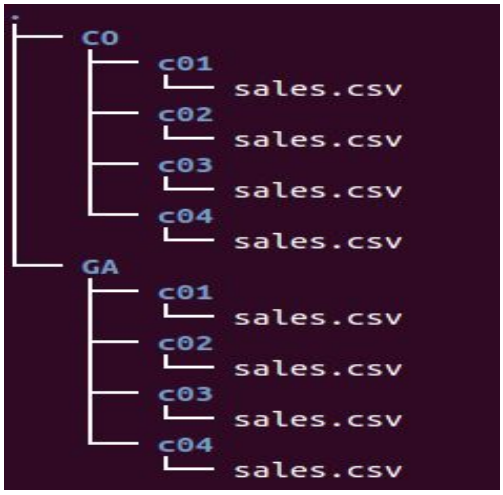
The partitioning levels should be read from a configuration file (see the next example) which dictates the order of partitioning.

Example:For this configuration file:

```
// content of config.txt
[meta]
data-path=/PATH_TO_MY_DATA

[hierarchy-data]
product=product.csv
location=location.csv
calendar=calendar.csv

[fact-data]
sales=sales.csv

[partitioning-keys]
location=state
product=category
```

The folder hierarchy will have at the higher level of the tree folders named after the state ids (location hierarchy), then inside each state folder, there will be folders named after each one of the categories (product hierarchy), then inside each one of the category folders we will find a sales.csv file that will contain sales data for ‘item, store’s that map to the grandparent and parent folder respectively. **Note** that we’re not partitioning by any level of the calendar hierarchy

here because there is no config entry for the calendar hierarchy in the [partitioning-keys] section of the config. If there were one then we should further partition by the corresponding level.



Validation Time!!!

Level 5

Facts data are not always keyed by base levels of our hierarchy, for example a file can have State (Location hierarchy) as a key, still all partitions need to have access to the data that is relevant to them.
In this level, you should partition the given *holiday.csv, category_holiday.csv* and *weather.csv* files (take in account the keys of each file and use it for a proper partitioning)

Level 6

In the retail world (and pretty much in any world really :-)), new data is available every minute, every second, and we want our systems to take advantage of the knowledge that comes with it. Your goal in this level is to consume a new sales increment (see *sales_increment.csv*) and use it to update the folder tree structure you previously produced based on the initial *sales.csv* file. As part of the *sales_increment.csv* data you need to expect that you may receive newly available data (for example: new sales), or corrections (for example: after catching a manual mistake). When receiving corrections, you need to override the old value with the corrected one. When receiving new values, we need to append it the old generated sales files.

Validation Time!!!

Level 7

One of the main reasons we decided to partition our data, was to be able to run a heavy science optimization process on it. Since we're planning on running this process in parallel for each independent partition, the total execution time would be the time it took the longest job to complete (provided there was zero failures -:)). In this level we assume that the execution time is a function of the sales file size.
Your task in this level is really to come up with a **partitioning strategy P**, that gives the **best** distribution of partitioned sales files **in terms of file size**. So, the output of your program is a set (P_Key, L_key, C_Key), where P_key, L_Key and C_Key are keys in respectively, the product, location and calendar hierarchy. E.g (Category, City, Month).

Validation Time!!!

Assumptions and Constraints

- Use any programming language you are comfortable with.
- You can only use libraries that are part of the standard library of that programming language. For Java for example, you can only use classes that belong to the Java Standard Library documented here: https://docs.oracle.com/javase/7/docs/api/ (could be any version though).
- You can use the internet and browse forums for answers to already existing questions but you can not ask questions on the internet or seek help from people other than the members of your group.
- Your solution should be capable of processing a large sales data file (5Gb). We will give you small and actual sized versions of that file to make your testing go faster...