Database Engine Assignment

Problem Statement

As an initial step of building a Utility to get meaningful information out of our Raw data - **as a first step you should be able to parse (decipher) our question.**
In our computing terms, we call this a query. Our system should be able to interpret this and break it into several parts - so that necessary actions can be triggered to fetch the required information in the proper format.

STEP 1 - Deciphering the parts of the String (Query)

Please note that the End User interacting with this utility will give out SQL like instructions and would expect the system to respond with necessary information. The system perceives this as a String of characters and we should manipulate and break this string into appropriate Data Structures. For Instance,

Query: **SELECT winner, Count(winner) FROM ipl.csv WHERE win_by_runs > 100 GROUP BY winner ORDER BY winner;**

Sample Data:

winner	count (winner)
Kings XI Punjab	1
Kolkata Knight Riders	1
Rajasthan Royals	1
Royal Challengers Bangalore	3

Query: **SELECT city, winner FROM ipl.csv WHERE winner='Chennai Super Kings' AND city ='Abu Dhabi' AND season=2014;**

Sample Data:

city	winner
Abu Dhabi	Chennai Super Kings

Few terms need to understand before starting the project.

1. select -> get the required information

2. '*' -> select all the fields

3. where -> filter result

4. order by -> sort the result based on particular field

5. group by -> get the records together based on particular field.

6. aggregate -> There are 5 aggregates,they are "sum,count,min,max,avg".

7. Example query string : select season,winner,player_match from ipl.csv where season > 2014 and city = 'Bangalore';

    a. fetching season, winner, player_match

    b. from the file ipl.csv (csv -> Comma Separated Value)

    c. where - filter the results. Get the details of the matches played in Bangalore after season 2014

    d. ipl.csv - name of the file from which we need to fetch the information/records

Our STEP 1 involves two tasks, given below:

a. Write a program to read the query string as input and split them into words. Print the output on console as given below:

Input String : select * from ipl.csv where season > 2014 and city = 'Bangalore'

Output String: select
*
from
ipl.csv
where
season
>
2014
and
city
=
'bangalore'

b. Further enhance your program to now extract certain parts of the same query:

i. Get only file name from the query string.

Input String : select * from ipl.csv where season > 2014 and city = 'Bangalore'
Output String : ipl.csv

ii. Get only base part(before `where` word) of the query from the given query string.

Input String : select * from ipl.csv where season > 2014 and city = 'Bangalore'
Output String : select * from ipl.csv

iii. Get only filter part(after `where` word) of the query from the given query string.

Input String : select * from ipl.csv where season > 2014 and city = 'Bangalore'
Output String : season > 2014 and city ='bangalore'

iv. As there will be multiple conditions, seperate each condition and display in different line.

Input String : select * from ipl.csv where season > 2014 and city = 'Bangalore'
Output String :
season > 2014
city ='bangalore'

v. Extract the logical operators in sequence from the given query string.
Note: Logical operators are "and, or, not"

Input String : select season,winner,player_match from ipl.csv where season > 2014 and city ='Bangalore' or date > '31-12-2014'
Output String :
and
or

vi. Extract the selected fields/information from the given query.

Input String : select city,winner,player_match from ipl.csv where season > 2014 and city = 'Bangalore'
Output String :
city
winner
player_match

vii. Extract the order by field from the given string.

Note : user may need the information in sorted order of a particular field.

Input String : select * from ipl.csv where season > 2016 and city= 'Bangalore' order by win_by_runs

Output String : win_by_runs

viii. Extract the group by field from the given string.

Note : user may need the related information grouped together.

For Example they may require to see the information department wise.

Input String : select team1, sum(win_by_runs) from ipl.csv where season > 2016 and city= 'Bangalore' group by team1

Output String : team1

ix. User may required the information like who is getting maximum salary or minimum age etc.. these are called aggregate functions (minimum, maximum, count, average, sum)

Input String : select avg(win_by_wickets),min(win_by_runs) from ipl.csv

Output String :

avg(win_by_wickets)

min(win_by_runs)

Note: Other parts like where clause, order by, group by may be present in the query.

Expected solution

Displaying various components/parts of the query like selected fields, conditional parts, aggregate fields, groupBy field, OrderBy field

Step 2 - Goals

In the first step, you worked on simply extracting different parts of the query and displaying these on the console. You programmed using a set of statements in Java.

Since then, you have spent time understanding OO concepts and it is now time to kick our database engine up one level. In a real engine, the first thing we would do is to create an object to hold the query. Now recall that queries are quite different - after all they are user defined. They can have different structures, even

though you are forcing a common language. So, for example, if there is a where clause, its contents are quite different from say a group-by clause.

What we will do in this step is to bring our OO thinking to the fore and design a set of objects to hold the various parts of a query.

Problem Statement

In our first step we extracted different parts of the query and displayed these on screen.

In this step, we push this further and create an OO way of holding query parameters. To do this we have to create a separate class with properties and methods.

The first thing to do is to give the class a meaningful name like QueryParameter. In this QueryParameterclass, we will add properties that store the various parts of the query string.

We will also create a bunch of other classes AggregateFunctions, and Restrictions to store parts of the query and queryParser to do the actual querying work.

Step 3 - Goals

In step 2, you split a query and created a bunch of classes to hold the parts of the query, along with the needed methods to access these properties. Now let us start working with data.

In this step, we will extend the previous assignment step and identify the header as well as datatype of each field and display the output in console.

Now, why would this be difficult? Well two reasons - the header row only contains names of the fields, not the data types. in fact the only way you can figure the data type of a column is to read a row (the first row works) and then each column in that row and then determine the data type. So some interesting code to do.

Problem Statement

Read the header

The CSV file contains a set of data, where the first line is header, for example: in our ipl.csv the following are header

```
id, season, city, date, team1, team2, toss_winner, toss_decision, result, dl_applied, winner,
win_by_runs, win_by_wickets, player_of_match, venue, umpire1, umpire2, umpire3
```

Identify the Datatype of each field

We cannot identify the data type of the field just by reading the header. We need to read a row which contains actual data.

All the data in the CSV file are in string format. We need to convert the non string data into its respective data type, this is needed because when ever we perform some sort of conditional operations we will be in need of the respective data type.

For example the following is the single line data of the CSV file

```
1, 2008, Bangalore, 2008-04-18, Kolkata Knight Riders, Royal Challengers Bangalore, Royal Challengers Bangalore, field, normal, 0, Kolkata Knight Riders, 140, 0, BB McCullum, M Chinnaswamy Stadium, Asad Rauf, RE Koertzen
```

Here you can easily find data which is not a string (for example 2008, 2008-04-18, 140 etc.,). The numeric data should be converted into integer or double.

However, at this stage, just to manage complexity, we will not try to convert date. It will be addressed in the next step.

Step 4 - Goals

In the previous step, you figured how to identify the datatype of individual columns but left date alone.

However, now that you have done some work with regExp, let us get the date parsing out of the way.

Problem Statement

In this step, we will try to convert all date type columns into their respective date pattern using regular expression.

Step 5 - Goals

In building our database engine, as of now, we can:

1. Parse a user defined query and construct an object representation for the query

2. Given a CSV file, identify headers and the data type of each column

In this step, we will begin executing queries - we will filter data from the CSV file based on to the query with first select fields and then multiple where conditions.

We will store the extracted data in a separate JSON file.

Sounds simple right. However there is a big catch here. Something that will trip you up. It is something to do with dynamic programming. Figured out what it is?

Problem Statement

In this assignment step, we will perform the following task:

1. Filter the data based on the query with multiple where conditions.

   For Example: where season >= 2013 or toss_decision!=bat and city=Bangalore;

2. Filter the data based on selected fields.

   For Example : select city,winner,team1,team2 from ipl.csv;

3. Store the filtered data in a JSON file.

Step 6 - Goals

In our previous step, we had filtered the CSV file based on multiple where condition and selected fields and stored the result in a JSON file

In this step, we will try to filter the data from CSV file with more complex queries using aggregate functions, group by, order by, aggregate group by. This step is going to test your understanding of collections, specifically keys and maps.

Problem Statement

In our previous assignment step, we had filtered the CSV file based on to multiple where condition and selected fields.

In this assignment step, we will try to filter the data from CSV file with much more complex queries. Our query is expected to contain aggregate functions, group by, order by, aggregate group by.

For example:

```
select city,winner,team1,team2 from ipl.csv where city='Bangalore' order by winner

select winner,count(*) from ipl.csv group by winner;

select city,sum(win_by_runs) from ipl.csv group by city

select count(city),sum(win_by_runs),min(season),max(win_by_wickets) from ipl.csv
```

Finally, store the filtered data in JSON file.

Step 7 - Goals

Remember the catch you faced in step 6 and all the code you wrote to handle the various types of expressions. Well let us refactor all of that now. Let us re-implement step 6, to filter the data from CSV file but use Lambda Expressions instead.

This is the last step of this course. Hope you had fun and learned a lot.

Problem Statement

Re-implement the query execution logic using lambda expressions to handle various cases like "where score > 23" or " team equals 'mumbai' " etc.