CSE 414 Homework 6: JSON, NoSQL, and AsterixDB

Objectives: To write queries over the semi-structured data model. To manipulate semi-structured data in JSON and use a NoSQL database system (AsterixDB).

Assigned date: November 22nd, 2020.

Due date: December 2nd, 2020 at 11 PM. You have 1.5 weeks to finish this assignment.

What to turn in:

A single file for each question, i.e., q1.sqlp, q2.sqlp, etc. It should contain commands executable by SQL++, and any text answers should be comments in the file (start comment lines with -- as in SQL).

Resources

- data: which contains mondial.adm (the entire dataset), country, mountain, and sea (three subsets)
- documentation for AsterixDB
- guide written by former TAs

Assignment Details

In this homework, you will write SQL++ queries over the semi-structured data model implemented in AsterixDB. Asterix is an Apache project on building a DBMS over data stored in JSON or ADM files.

Mondial Dataset

You will run queries over the Mondial database, a geographical dataset aggregated from multiple sources. As is common in real-world aggregated data, the Mondial dataset is *messy*; the schema is occasionally inconsistent, and some facts may conflict. We have provided the dataset in ADM format, converted from the XML format available online, for use in AsterixDB.

Setting up AsterixDB

1. Download and install AsterixDB. Download the file apache-asterixdb-0.9.4.zip and unzip it anywhere you'd like.

2. (extra step for some users:) You need to install the **Java 8 SE Development Kit** if you don't already have it. Follow this link to the download for your system. Many people will have this already if you've installed the tools for writing Java programs. On Windows, just installing the JDK should be enough. For Mac you may need to do some additional steps as described here.
3. Start an instance of AsterixDB using the **start-sample-cluster.sh** (or **.bat** if you are on Windows) located in the **opt/local/bin** folder.
4. When your AsterixDB instance is running you can enter the query interface by visiting 127.0.0.1:19001 in your favorite web browser. It may take a few minutes before you can access the page, while the instance starts up.
5. Download the geographical data from the link in the resources above. The data are JSON data files; you can inspect them using your favorite text editor.
6. Create a dataverse of Mondial data. Copy and paste the text below in the Query box of the web interface. Edit the <path to mondial.adm>. Then press Run:

```
 DROP DATAVERSE geo IF EXISTS;
 CREATE DATAVERSE geo;

 CREATE TYPE geo.worldType AS {auto_id:uuid };
 CREATE DATASET geo.world(worldType)  PRIMARY KEY auto_id AUTOGENERATED;
 LOAD DATASET geo.world USING localfs
        (("path"="127.0.0.1:///<path to mondial.adm>, e.g.,
localhost://C:/Users/XXX/Desktop/hw7/data/mondial.adm"),("format"="adm"));
 /* Edit the absolute path above to point to your copy of mondial.adm. */
 /* Use '\' instead of '/' in a path for Windows. e.g.,
C:\414\hw\geo\mondial.adm. */
```

7. Alternatively, you can use the terminal to run queries rather than the web interface. After you have started Asterix, put your query in a file (say **q1.sqlp**), then execute the query by typing the following command in terminal:

```
curl -v --data-urlencode "statement=`cat q1.sqlp`" --data pretty=true
http://localhost:19002/query/service
```

This will print the output on the screen. If there is too much output, you can save it to a file

```
curl -v --data-urlencode "statement=`cat q1.sqlp`" --data pretty=true
http://localhost:19002/query/service  > output.txt
```

You can now view **output.txt** using your favorite text editor.

8. To reference the geodatabase, use the statement USE geo; before each of your queries to declare the geo namespace. Alternatively, prefix every dataset with geo. Try this query to see if things are running correctly:

```
SELECT y.`-car_code` as code, y.name as name
FROM geo.world x, x.mondial.country y
ORDER BY y.name;
```

9. For practice, run, examine, modify these queries. They contain useful templates for the questions on the homework: make sure you understand them.

```
-- return the set of countries
SELECT x.mondial.country FROM geo.world x;

-- return each country, one by one (see the difference?)
SELECT y as country FROM geo.world x, x.mondial.country y;

-- return just their codes, and their names, alphabetically
-- notice that -car_code is not a legal field name, so we enclose in ` ...
`
SELECT y.`-car_code` as code, y.name as name
FROM geo.world x, x.mondial.country y order by y.name;

-- this query will NOT run...
SELECT z.name as province_name, u.name as city_name
FROM geo.world x, x.mondial.country y, y.province z, z.city u
WHERE  y.name='Hungary';
-- ...because some provinces have a single city, others have a list of
cities; fix it:

SELECT z.name as province_name, u.name as city_name
FROM geo.world x, x.mondial.country y, y.province z,
            CASE  WHEN is_array(z.city) THEN z.city
                  ELSE [z.city] END u
WHERE  y.name='Hungary';

-- same, but return the city names as a nested collection;
-- note correct treatment of missing cities
-- also note the convenient LET construct (see SQL++ documentation)
SELECT z.name as province_name, (select u.name from cities u) as cities
FROM geo.world x, x.mondial.country y, y.province z
LET cities = CASE  WHEN z.city is missing THEN []
                   WHEN is_array(z.city) THEN z.city
                   ELSE [z.city] END
WHERE  y.name='Hungary';
```

10. To shutdown Asterix, simply run **stop-sample-cluster.sh** in the terminal. The script is located in **opt/local/bin** (or **opt\local\bin\stop-sample-cluster.bat** on windows).

Problems (100 points)

For all questions asking to report free response-type questions, please leave your responses in comments

Use only the mondial.adm dataset for problems 1-9.

1. Retrieve the names of all cities located in Peru, sorted alphabetically. Name your output attribute **city**. [Result Size: 30 rows of **{"city":...}**]

2. For each country return its name, its population, and the number of religions sorted alphabetically by country. Report 0 religions for countries without religions. Name your output attributes **country**, **population**, **num_religions**. [Result Size: 238 rows of **{"num_religions":..., "country":..., "population":...}** (order of keys can differ)]

3. For each religion return the number of countries where it occurs; order them in decreasing number of countries. Name your output attributes **religion**, **num_countries**. [Result size: 37 of **{"religion':..., "num_countries":...}** (order of keys can differ)]

4. For each ethnic group, return the number of countries where it occurs, as well as the total population world-wide of that group. Hint: you need to multiply the ethnicity's percentage with the country's population. Use the functions **float(x)** and/or **int(x)** to convert a **string** to a **float** or to an **int**. Name your output attributes **ethnic_group**, **num_countries**, **total_population**. You can leave your final **total_population** as a **float** if you like. [Result Size: 262 of **{"ethnic_group":..., "num_countries":..., "total_population":...}** (order of keys can differ)]

5. Compute the list of all mountains, their heights, and the countries where they are located. Here you will join the "mountain" collection with the "country" collection, on the country code. You should return a list consisting of the mountain name, its height, the country code, and country name, in descending order of the height. Name your output attributes **mountain**, **height**, **country_code**, **country_name**. [Result Size: 272 rows of **{"mountain":..., "height":..., "country_code":..., "country_name":...}** (order of keys can differ)]

   Hint: Some mountains can be located in more than one country. You need to output them for each country they are located in.

6. Compute a list of countries with all their mountains. This is similar to the previous problem, but now you will group the mountains for each country; return both the mountain name and its height. Your query should return a list where each element

consists of the country code, country name, and a list of mountain names and heights; order the countries by the number of mountains they contain, in descending order. Name your output attributes **country_code**, **country_name**, **mountains**. The attribute **mountains** should be a list of objects, each with the attributes **mountain** and **height**. [Result Size: 238 rows of **{"country_code":..., "country_name":..., "mountains": [{"mountain":..., "height":...}, {"mountain":..., "height":...}, ...]}** (order of keys can differ)]

7. Find all countries bordering two or more seas. Here you need to join the "sea" collection with the "country" collection. For each country in your list, return its code, its name, and the list of bordering seas, in decreasing order of the number of seas. Name your output attributes **country_code**, **country_name**, **seas**. The attribute **seas** should be a list of objects, each with the attribute sea. [Result Size: 74 rows of **{"country_code":..., "country_name":..., "seas": [{"sea":...}, {"sea":...}, ...]}** (order of keys can differ)]

8. Return all landlocked countries. A country is landlocked if it borders no sea. For each country in your list, return its code, its name, in decreasing order of the country's area. Note: this should be an easy query to derive from the previous one. Name your output attributes **country_code**, **country_name**, **area**. [Result Size: 45 rows of **{"country_code":..., "country_name":..., "area":...}** (order of keys can differ)]

9. For this query you should also measure and report the runtime; it may be approximate (warning: it might run for a while). Find all distinct pairs of countries that share both a mountain and a sea. Your query should return a list of pairs of country names. Avoid including a country with itself, like in (France,France), and avoid listing both (France,Korea) and (Korea,France) (not a real answer). Name your output attributes **first_country**, **second_country**. [Result Size: 7 rows of **{"first_country":..., "second_country":...}]**

For problems 10-12 we will ask you to load in the extra datasets provided in the data folder.

10. Create a new dataverse called geoindex, then run the following commands:

```
USE geoindex;
CREATE TYPE countryType AS OPEN {
    `-car_code`: string,
    `-area`: string,
    population: string
};
CREATE DATASET country(countryType)
   PRIMARY KEY `-car_code`;
CREATE INDEX countryID ON country(`-car_code`) TYPE BTREE;
LOAD DATASET country USING localfs
```

```
    (("path"="127.0.0.1://<path to country.adm>, e.g.,
/hw7/data/country.adm"),("format"="adm"));
```

This created the type **countryType**, the dataset **country**, and a **BTREE** index on the attribute **-car_code**, which is also the primary key. Both types are OPEN, which means that they may have other fields besides the three required fields **-car_code**, **-area**, and population.

Create two new types: **mountainType** and **seaType**, and two new datasets, **mountain**, and **sea**. Both should have two required fields: **-id** and **-country**. Their key should be autogenerated, and of type **uuid** (see how we did it for the mondial dataset). Create an index of type **KEYWORD** (instead of **BTREE**) on the **-country** field (for both **mountain** and **sea**). Turn in the complete sequence of commands for creating the dataverse and all three types, datasets, and indices (for **country**, **mountain**, **sea**).

Recall from the lecture that Asterix only allows creating index at the top-level collection, hence we provide the country, sea, etc collections individually even though their data is already included in mondial.

11. Re-run the query from 9. (“pairs of countries that share both a mountain and a sea”) on the new dataverse geoindex. Turn in your altered query and report on the new runtime. [Result Size: 7 rows of **{"first_country":..., "second_country":...}]**

12. Modify the query from 11. to return, for each pair of countries, the list of common mountains, and the list of common seas. Name your output attributes **first_country**, **second_country**, **mountains**, **seas**. [Result Size: 7 rows of **{"mountains":[{"mountain":...}, ...], "seas":[{"sea":...}, ...], "first_country":..., "second_country":...}]**

Submission Instructions

Write your answers in a file for each question: **q1.sqlp, … , q12.sqlp** to Gradescope. Leave your runtime and other responses in comments.