DATA 514 Homework 5: JSON, NoSQL, and AsterixDB

Objectives: To practice writing queries over the semi-structured data model. To be able to manipulate semistructured data in JSON and practice using a NoSQL database system (AsterixDB).

Assigned date: Tuesday, Feb 26, 2019 (but we will release it earlier)

Due date: Monday, March. 4, 2019. You have 1 week for this homework.

What to turn in:

A single file for each question, i.e., `hw5-q1.sqlp`, `hw5-q2.sqlp` etc in the `submission` directory. It should contain commands executable by SQL++, and should contain comments for text answers (delimited by `--` as in SQL).

Resources

- Starter code: which contains `monidal.adm` (the entire dataset), `country`, `mountain`, and `sea` (three subsets)
- documentation for AsterixDB
- mailing list for AsterixDB / SQL++ quetions. Sign up on the "users" mailing list and you can post questions subsequently (you can unsubscribe afterwards if you like).

Assignment Details

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

A. Setting up AsterixDB (0 points)

Asterix is already installed in the CSE Windows lab machines and the VDI machines (in `c:\asterixdb`). If you prefer to run it on your own devices, follow step 1 below.

1. Download and install AsterixDB (optional):

- Go to the Asterix download website and download the "Simple Server Package." This downloads the stable version of Asterix (0.9.2). Unzip the downloaded file.

2. Download the assignment files and uncompress them. All of them are JSON data files, you can inspect them using your favorite text editor.

3. Start the server. Go to the Asterix documentation website and follow the instructions listed under "Option 1: Using NC Services." Follow the instructions under “Quick Start”. You are done once you can open the web interface in your browser. In the web interface, select:

    - Query language SQL++
    - Output format JSON (lossless)

4. Copy, paste, and edit the red text in the Query box, the press Run:

```
DROP DATAVERSE hw5 IF EXISTS;
CREATE DATAVERSE hw5;
USE hw5;
CREATE TYPE worldType AS {auto_id:uuid };
CREATE DATASET world(worldType)  PRIMARY KEY auto_id AUTOGENERATED;
LOAD DATASET world USING localfs
        (("path"="127.0.0.1:///<path to mondial.adm>, e.g.,
/514/hw5/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:/514/hw5/mondial.adm. */
```

```
/* Note: if you type one command at a time, then end it with a ";" */
USE hw5;
```

5. 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
USE hw5;
SELECT x.mondial.country FROM world x;
```

```
-- return each country, one by one (see the difference?)
USE hw5;
SELECT y as country FROM 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 `
… `
USE hw5;
SELECT y.`-car_code` as code, y.name as name
FROM world x, x.mondial.country y order by y.name;
```

```
-- this query will NOT run...
USE hw5;
SELECT z.name as province_name, u.name as city_name
FROM 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:
```

```
USE hw5;
SELECT z.name as province_name, u.name as city_name
FROM 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)

USE hw5;
SELECT z.name as province_name, (select u.name from cities u)
FROM 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';
```

6. To shutdown Asterix, simply run `opt/local/bin/stop-sample-cluster.sh` (or opt\local\bin\stop-sample-cluster.bat` on windows).

B. Problems (100 points)

1. (5 points) Retrieve all the names of all cities located in Peru, sorted alphabetically. [Result Size: 30 rows]
2. (10 points) For each country return its name, its population, and the number of religions, sorted alphabetically by country. [Result Size: 238 rows]
3. (10 points) For each religion return the number of countries where it occurs; order them in decreasing number of countries. [Result size: 37]
4. (10 points) 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 x to a float or to an int. [Result Size: 262]
5. (10 points) 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. [Result Size: 272 rows]
6. (10 points) Compute a list of countries with all their mountains. This is similar to the previous problem, but now you will group the moutains 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 [Result Size: 238]

7. (10 points) 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 lits list of bordering seas, in decreasing order of the number of seas. [Result Size: 74]

8. (10 points) Return all landlocked countries. A country is landlocked if it borders no sea. Order your answers in decreasing order of the country's area. Note: this should be an easy query to derive from the previous one. [Result Size: 45]

9. (10 points) For this query you should also measure and report the runtime; it may be approximate (expect it around 10'-30') . 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). [Result Size: 7]

10. (13 points) Create a new dataverse called hw5index, then run the following commands:

```
USE hw5index;
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.,
/514/hw5/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 `moutnain` and `sea`). Turn in the complete sequence of commands for creating all three types, datasets, and indices (for `country`, `mountain`, `sea`).

11. (1 points) Re-run the query from 9. (“pairs of countries that share both a mountain and a sea”) on the new dataverse `hw5index`. Report the new runtime. [Result Size: 7]

12. (1 points) Modify the query from 11. to return, for each pair of countries, the list of common mountains, and the list of common seas. [Result Size: 7]

Submission Instructions

Write your answers in file `hw5-q1.sqlp`, and `hw5-q2.sqlp`, etc.