CSE 344 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: Wednesday, April 25, 2018.

Due date: Wednesday, May 2, 2018.

Questions: Post them on the discussion board. Tag your post with "Asterix" on Piazza if it pertains to the syntax of SQL++ / AsterixDB installation problems. Otherwise tag your questions with "HW5."

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).
- AsterixDB draft tutorial. You need to log in with your UW net ID to access this document. As mentioned in class, this is still work in progress. The Asterix folks have asked us not to share this with others outside of this class.

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)

1. Download and install AsterixDB in your home VM:

- Download this zip file and unzip it somewhere in your home directory. *Note: If you are installing it on your own machine, note that AsterixDB requires Java 8; Java 9 will not work.*

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

   If you use the home VM, `start-sample-cluster.sh` is located in `<directory that you unzipped the file above>/opt/local/bin`. You can start Asterix by first going to the `bin` directory and then run `JAVA_HOME=/ ./start-sample-cluster.sh` (or `start-sample-cluster.bat` on windows, you might see a few extra windows pop up when it starts, you can ignore those).

   Running the script might seemingly perform nothing. But if it works then you should be able to open the web interface in your browser, by visiting `http://localhost:19001` as described in the website. In the web interface, select:

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

3. Copy, paste, and edit the `<path to modial.adm` text in the Query box, then press Run:
   `sql DROP DATAVERSE hw5 IF EXISTS; CREATE DATAVERSE hw5; USE hw5; CREATE TYPE worldType AS {auto_id:uuid }; CREATE DATASET world(worldType`
   `sql /* Note: if you type one command at a time, then end it with a ";" */ USE hw5;`
4. Alternatively, you can also use the terminal to run queries rather than the web interface. After you have started Asterix, put your query in a file (say `hw5-q1.sqlp`), then execute the query by typing the following command in terminal:

   ```
   curl -v --data-urlencode "statement=`cat hw5-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 hw5-q1.sqlp`" --data pretty=true http://localhost:19002/query/service  >  output.txt
   ```

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

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 `stop-sample-cluster.sh` in the terminal. If you are using the home VM, this script is located in `<directory that you unzipped the file above>/opt/local/bin` (or `opt\local\bin\stop-sample-cluster.bat` on windows). If you are using the VM, go to the `bin` directory and then run `JAVA_HOME=/ ./stop-sample-cluster.sh` to shut down Asterix.

B. Problems (100 points)

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

Use only the `mondial` dataset for problems 1-9. For problems 10-12 we will ask you to load in extra datasets provided in starter code.

1. Retrieve all the names of all cities located in Peru, sorted alphabetically. Name your output attribute `cities`. [Result Size: 30 rows]
2. For each country return its name, its population, and the number of religions, sorted alphabetically by country. Name your output attributes `country`, `population`, `num_religions`. [Result Size: 238 rows]
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]
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]
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]

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`, `mountain`, `mountain_height`. [Result Size: 238]

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`, `sea`. [Result Size: 74]

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]

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]

10. Create a new dataverse called hw5index, then run the following commands:
`sql USE hw5index; CREATE TYPE countryType AS OPEN { `-car_code`: string, `-area`: string, population: string }; CREATE DATASET country(co`

    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 all three types, datasets, and indices (for `country`, `mountain`, `sea`).

    Recall from lecture that asterix only allows creating index at 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 `hw5index`. Report the new runtime. [Result Size: 7]

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`, `mountain`, `sea`. [Result Size: 7 rows of `{"mountains":[{"mountain":...}, {"mountain":...}], "seas":[{"sea":...}, {"sea":...}], {"first_country":..., "second_country":...}`]

Submission Instructions

Write your answers in a file for each question: `hw5-q1.sqlp`, ... `hw5-q12.sqlp`. Leave your runtime and other responses in comments.

Important: To remind you, in order for your answers to be added to the git repo, you need to explicitly add each file:

```
$ git add hw5-q1.sqlp ...
```

Again, just because your code has been committed on your local machine does not mean that it has been submitted -- it needs to be on GitLab!

Use the same bash script `turnIn_Hw5.sh` in the root level directory of your repository that commits your changes, deletes any prior tag for the current lab, tags the current commit, and pushes the branch and tag to GitLab.

If you are using the home VM or Mac OSX, you should be able to run the following:

```
$ ./turnIn_Hw5.sh
```

Like previous assignments, make sure you check the results afterwards to make sure that your file(s) have been committed.