Section 7 Worksheet Solutions

Some keywords and functions:

`is_array( … )`	checks if value is an array
`split(s, d)`	splits string s on delimiter d
`[ … ]`	explicitly construct array
`(CASE WHEN … THEN … ELSE … END)`	combine with "is_array(…)"
`MISSING`	reserved keyword like "NULL"
`` ` … ` ``	back tick needed for accessing keys with names containing "-", "#", etc.
`SELECT VALUE …`	used to unnest objects

Use "`mondial.adm`" as the data set for the following questions. Assume you have the dataverse "`hw5`", and the dataset "`world`" like was given in homework 5.

1. Find the highest population city in France. Give the city name and the population.

The initial challenge of using SQL++ after learning SQL is not stuff like subqueries but rather how to go about accessing the data you want. This challenge, of course, comes from the non-flat/semi-structured format of the data we are accessing (in this case JSON).

The idea when grabbing hierarchical data is to "traverse" down the hierarchy. This, perhaps, makes the FROM clause in query the most important part. With our imported data, we observe that cities are located in provinces which are located in countries which are located in the world. Translating that to our FROM clause, we roughly get "`world AS X, X.mondial.country Y, Y.province AS Z, Z.city AS U`".

However now we need to deal with the other hard part of semi-structured data which is heterogeneity. We solve this with CASE statements as needed. Remember that SQL++ only accepts collections in the FROM clause, so we need to rectify single objects to singleton array values.

The rest of the query uses concepts that were covered in normal SQL. Using the below query, we get "Paris" as the highest population city (as would be expected).

```
SELECT U.name AS city_name,
       int(U.population.`#text`) AS population
  FROM world AS X,
       X.mondial.country AS Y,
       Y.province AS Z,
       (CASE WHEN is_array(Z.city)
             THEN Z.city
             ELSE [Z.city] END) AS U
 WHERE Y.name = 'France' AND
       U.population IS NOT MISSING
 ORDER BY int(U.population.`#text`) DESC
 LIMIT 1;
```

2. List all deserts and which countries they lie in. Return the name of the desert and an array of the names of the countries.

There are many ways to go about solving queries like this.

The first solution is tries to split this problem into subproblems. We can grab pairs of corresponding deserts and countries with a WITH subquery. After we can conglomerate them into an array using LET. The second solution is more direct, creating the array of countries for each desert.

These solutions both take advantage of the LET clause in SQL++ which can create values correlated to items in the FROM clause (very much like a SELECT subquery, but more readable).

```
-- example solution 1 (~1 sec)
WITH tmp AS (SELECT y.name AS desert,
                    x.name AS country
               FROM world w,
                    w.mondial.country x,
                    w.mondial.desert y,
                    split(y.`-country`, ' ') z
              WHERE x.`-car_code` = z)
SELECT t.desert AS desert,
       c as countries
  FROM tmp t
   LET c = (SELECT VALUE r.country
              FROM tmp r
             WHERE r.desert=t.desert);

-- example solution 2 (~30 sec)
SELECT y.name AS desert, z AS countries
  FROM world w, w.mondial.desert y
   LET z = (SELECT VALUE x.name
              FROM split(y.`-country`, ' ') AS z,
                   w.mondial.country AS x
             WHERE x.`-car_code` = z);
```