Report on Week 4

Lei Liu, 9669373

1. Task 1

1.1 Return names of countries that are not landlocked (i.e., have a sea coast).

select distinct c.name from country c join geo\_sea s on c.code = s.country;

1.2 Return the names of all lakes, rivers and seas.

select name from lake UNION select name from river UNION select name from sea;

1.3 Return the average length over all the river.

select avg(length) from river;

1.4 Return the name of countries that have more than 10 islands.

select c.name from country c join geo\_island i on c.code=i.country group by i.country having count(i.country) > 10;

1.5 Return, for every river in Great Britian, the length of that river.

select r.name, r.length from geo\_river gr join river r on gr.river=r.name where gr.country='GB';

1.6 Return the name of the countries that have the 10 longest total length of rivers.

select name, sum(length) as total\_length from (select distinct c.name, r.length from country c, geo\_river gr, river r where c.code=gr.country and gr.river = r.name) group by name order by total\_length desc limit 10;

1.7 Return all the information available about cities whose name is Manchester.

select \* from City where name='Manchester';

1.8 Return the name of cities whose name starts with the substring 'Man’.

select name from city where name like 'Man%';

1.9 Return the name of the country and the name of the organization of countries with Buddhist populations that are members of organizations established after 1st December 1994.

select c.name as country, o.name from country c, organization o, ismember im, religion re where im.organization = o.abbreviation and c.code = im.country and re.country=c.code and o.established > '1994-12-01' and re.name = 'Buddhist';

1.10 Return the name of each country with the number of islands in it.

select c.name, count(geo.island) as num\_island from country c join geo\_island geo on c.code=geo.country group by geo.island;

2. Task 2

XQueries in task 2 is base on one file-mondial.xml.

2.1 Return the names of countries that are not landlocked (i.e., have a sea coast).

distinct-values(

for $c in doc('xml/mondial.xml')//country

let $code := $c/@car\_code

for $geo\_sea in doc('xml/mondial.xml')//sea/tokenize(@country)

where $code = $geo\_sea

return $c/name )

2.2 Return the names of all lakes, rivers and seas.

let $l := doc('xml/mondial.xml')//lake/name/text()

let $r := doc('xml/mondial.xml')//river/name/text()

let $s := doc('xml/mondial.xml')//sea/name/text()

return $l union $r union $s

2.3 Return the average length over all the rivers in the database.

let $l := doc('xml/mondial.xml')//river/length/text()

return avg($l)

2.4 Return the name of countries that have more than 10 islands.

for $c in doc('xml/mondial.xml')//country

where count(doc('xml/mondial.xml')//island[tokenize(@country)=$c/@car\_code]/@id ) > 10

return $c/name/text()

2.5 Return, for every river in Great Britain, the length of that river.

for $r in doc('xml/mondial.xml')//river[tokenize(@country)='GB']

return $r/length

2.6 Return the name of the countries that have the 10 longest total length of rivers.

(for $c in doc('xml/mondial.xml')//country

let $l := sum(doc('xml/mondial.xml')//river[tokenize(@country)=$c/@car\_code]/length/text() )

order by $l descending

return concat($c/name, " ", $l)

)[position()<=10]

2.7 Return all the information available about cities whose name is Manchester.

doc('xml/mondial.xml')//city[name='Manchester']

2.8 Return the name of cities whose name starts with the substring 'Man’.

for $cn in doc('xml/mondial.xml')//city/name/text()

where starts-with( $cn, 'Man')

return $cn

2.9 Return the name of the country and the name of the organization of countries with Buddhist populations that are members of organizations established after 1st December 1994.

for $c in doc('xml/mondial.xml')//country[religion='Buddhist']

let $member := tokenize($c/@memberships)

for $o in doc('xml/mondial.xml')//organization[@id=$member and established > '1994-12-01' ]

return concat($c/name, " ---- ", $o/name)

2.10 Return the name of each country with the number of islands in it.

for $c in doc('xml/mondial.xml')//country

let $num := count(doc('xml/mondial.xml') //island[tokenize(@country)=$c/@car\_code] )

return concat($c/name, " ", $num)

3. Task 3

3.1 Return the names of countries that are not landlocked (i.e., have a sea coast).

distinct-values(

for $c in doc('xml/converted/a/country\_a.xml')//country-tuple

for $geo in doc('xml/converted/a/geo\_sea\_a.xml')//geo\_sea-tuple

where $c/@code = $geo/@country

return $c/@name)

3.2 Return the names of all lakes, rivers and seas.

let $l := doc('xml/converted/a/lake\_a.xml')//lake-tuple/@name

let $r := doc('xml/converted/a/river\_a.xml')//river-tuple/@name

let $s := doc('xml/converted/a/sea\_a.xml')//sea-tuple/@name

return $l union $r union $s

3.3 Return the average length over all the rivers in the database.

let $l := data( doc('xml/converted/a/river\_a.xml')//river-tuple[@length != 'MISSING']/@length )

return avg($l)

3.4 Return the name of countries that have more than 10 islands.

for $c in doc('xml/converted/a/country\_a.xml')//country-tuple

where count( doc('xml/converted/a/geo\_island\_a.xml')//geo\_island-tuple[@country=$c/@code] ) > 10

return $c/@name

3.5 Return, for every river in Great Britain, the length of that river.

for $gr in doc('xml/converted/a/geo\_river\_a.xml')//geo\_river-tuple[@country='GB']

for $r in doc('xml/converted/a/river\_a.xml')//river-tuple[@name=$gr/@river]

return concat($r/@name, " ", $r/@length)

3.6 Return the name of the countries that have the 10 longest total length of rivers.

(for $c in doc('xml/converted/a/country\_a.xml')//country-tuple

let $r := distinct-values( doc('xml/converted/a/geo\_river\_a.xml')//geo\_river-tuple[@country=$c/@code]/@river )

let $l := sum( data( doc('xml/converted/a/river\_a.xml')//river-tuple[@name=$r and @length != 'MISSING']/@length ) )

order by $l descending

return concat($c/@name, " ", $l)

)[position()<=10]

3.7 Return all the information available about cities whose name is Manchester.

doc('xml/converted/a/city\_a.xml')//city-tuple[@name='Manchester']

3.8 Return the name of cities whose name starts with the substring 'Man’.

for $cn in doc('xml/converted/a/city\_a.xml')//city-tuple/@name

where starts-with( $cn, 'Man')

return $cn

3.9 Return the name of the country and the name of the organization of countries with Buddhist populations that are members of organizations established after 1st December 1994.

for $r in doc('xml/converted/a/religion\_a.xml')//religion-tuple[@name='Buddhist']/@country

for $c in doc('xml/converted/a/country\_a.xml')//country-tuple[@code=$r]

let $member := doc('xml/converted/a/ismember\_a.xml')//ismember-tuple[@country=$c/@code]/@organization

for $o in doc('xml/converted/a/organization\_a.xml')//organization-tuple[@abbreviation=$member and @established > '1994-12-01']

return concat($c/@name, " ---- ", $o/@name)

3.10 Return the name of each country with the number of islands in it.

for $c in doc('xml/converted/a/country\_a.xml')//country-tuple

let $num := count( doc('xml/converted/a/geo\_island\_a.xml')//geo\_island-tuple[@country=$c/@code] )

return concat($c/@name, " ", $num)

4. Task 4

4.1 Return the names of countries that are not landlocked (i.e., have a sea coast).

distinct-values(

for $c in doc('xml/converted/e/country\_e.xml')//country-tuple

for $geo in doc('xml/converted/e/geo\_sea\_e.xml')//geo\_sea-tuple

where $c/code = $geo/country

return $c/name)

4.2 Return the names of all lakes, rivers and seas.

let $l := doc('xml/converted/e/lake\_e.xml')//lake-tuple/name

let $r := doc('xml/converted/e/river\_e.xml')//river-tuple/name

let $s := doc('xml/converted/e/sea\_e.xml')//sea-tuple/name

return $l union $r union $s

4.3 Return the average length over all the rivers in the database.

let $l := data( doc('xml/converted/e/river\_e.xml')//river-tuple[length != 'MISSING']/length )

return avg($l)

4.4 Return the name of countries that have more than 10 islands.

for $c in doc('xml/converted/e/country\_e.xml')//country-tuple

where count( doc('xml/converted/e/geo\_island\_e.xml')//geo\_island-tuple[country=$c/code] ) > 10

return $c/name

4.5 Return, for every river in Great Britain, the length of that river.

for $gr in doc('xml/converted/e/geo\_river\_e.xml')//geo\_river-tuple[country='GB']

for $r in doc('xml/converted/e/river\_e.xml')//river-tuple[name=$gr/river]

return concat($r/name, " ", $r/length)

4.6 Return the name of the countries that have the 10 longest total length of rivers.

(for $c in doc('xml/converted/e/country\_e.xml')//country-tuple

let $r := distinct-values( doc('xml/converted/e/geo\_river\_e.xml')//geo\_river-tuple[country=$c/code]/river )

let $l := sum( data( doc('xml/converted/e/river\_e.xml')//river-tuple[name=$r and length != 'MISSING']/length ) )

order by $l descending

return concat($c/name, " ", $l)

)[position()<=10]

4.7 Return all the information available about cities whose name is Manchester.

doc('xml/converted/e/city\_e.xml')//city-tuple[name='Manchester']

4.8 Return the name of cities whose name starts with the substring 'Man’.

for $cn in doc('xml/converted/e/city\_e.xml')//city-tuple/name

where starts-with( $cn, 'Man')

return $cn

4.9 Return the name of the country and the name of the organization of countries with Buddhist populations that are members of organizations established after 1st December 1994.

for $r in doc('xml/converted/e/religion\_e.xml')//religion-tuple[name='Buddhist']/country

for $c in doc('xml/converted/e/country\_e.xml')//country-tuple[code=$r]

let $member := doc('xml/converted/e/ismember\_e.xml')//ismember-tuple[country=$c/code]/organization

for $o in doc('xml/converted/e/organization\_e.xml')//organization-tuple[abbreviation=$member and established > '1994-12-01']

return concat($c/name, " ---- ", $o/name)

4.10 Return the name of each country with the number of islands in it.

for $c in doc('xml/converted/e/country\_e.xml')//country-tuple

let $num := count( doc('xml/converted/e/geo\_island\_e.xml')//geo\_island-tuple[country=$c/code] )

return concat($c/name, " ", $num)

5 Task 5

Figure 1. Statistic and comparison of average executing time of different queries on different kind of database

Table 1. Statistic of Query Result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SQL | Singe XML | Attribute XMLs | Element XMLs |
| Query 1 | 198 | 198 | 193 | 193 |
| Query 2 | 413 | 432 | 382 | 382 |
| Query 3 | 1 | 1 | 1 | 1 |
| Query 4 | 12 | 9 | 11 | 11 |
| Query 5 | 4 | 1 | 9 | 9 |
| Query 6 | 10 | 10 | 10 | 10 |
| Query 7 | 2 | 2 | 2 | 2 |
| Query 8 | 18 | 20 | 16 | 16 |
| Query 9 | 177 | 177 | 18 | 18 |
| Query 10 | 293 | 244 | 238 | 238 |

In figure 1, it is clear that all SQLs cost the least time than other xqueries, around 1 to 2 milliseconds. While, the cost of Xqueries on a single xml file were significantly huge, all above 200ms. On xml model with attribute and element, the xqueries’ executing time were very similar, ranged from 20 to 80 milliseconds.

From the observation above, we could see that SQL queries on sqlite3 performed best. The reason could be a) good performance of sqlite3 DBMS than other BaseX, such as using index in sqlie3 b) better optimizer of sqlite3 than others’, c) for this experiment, data structured in table model was much suitable than in tree model. The file this relational data is 2.6MB.

Querying on a single file, which is a 3.5MB xml file, was the most costly. The results between this with SQL queries were barely different (less than 5% difference), as shown in table 1. The reason could be that a great amount of memory and time are needed on loading file and searching and filtering data from the whole file.

Comparing the cost between querying on attribute XMLS with element XMLs, they produced the same result from same data, yet, querying on attribute XMLs were slightly slower. The size of attribute and element XML files are 1.2MB and 5.2 MB respectively. They have different size of database file, but similar querying time. The reason could be that when querying, the numbers of accessed nodes are likely similar, thus cost similar time. However, it seems xml using elements could be better for query.

In conclusion, using relational DBMS is better than using XML database, the size of database file is moderate, and can be queried in a fast approach. Among three different XML model, using a single file is worst, resulting in large size of intermediate data and cost huge time on query. The XML model based on attribute requires the smallest size of file, but its performance on query needs improvement. Although the largest file is needed for modelling XML with elements, query on this performs much better than a single XML or with attribute. Thus, when modelling an xml database, it is better to model data with element or attribute.