XQuery 1: Listo të gjitha kryeqytetet në të cilat nuk kalon asnjë lum, mirëpo janë anëtare të paktën të një organizate.

```
for $country in //country
where string-length(normalize-space($country/@memberships)) > 0
let $capitalId := $country/@capital
let $capitalCity := //city[@id = $capitalId]
where not($capitalCity/located_at[@watertype = "river"])
let $capitalName := $capitalCity/name[1]
return
  <capital>{string($capitalName)}</capital>
```

XQuery 2: Listo lumenjtë të cilët rrjedhin nëpër shtete të cilat nuk janë anëtarë të NATO-s, kanë dalje në dete si dhe numri i luemnjëve në këto shtete është më i madh se 10.

```
let $seaCountries := distinct-values(
  for $sea in //sea
  return tokenize($sea/@country, '\s+')
)

(: Build map of countries that meet all 3 conditions :)
let $qualifiedMap :=
  for $country in //country
  let $code := $country/@car_code
  where $code = $seaCountries
    and not(contains($country/@memberships, "org-NATO"))
  let $countryRivers :=
    distinct-values(
      for $river in //river
      where some $loc in $river/located satisfies $loc/@country = $code
      return $river/@id
    )
  where count($countryRivers) > 10
  return <qualified code="{$code}">{
    for $id in $countryRivers
    return <riverId>{ $id }</riverId>
  }</qualified>

(: Now use $qualifiedMap :)
return
  for $q in $qualifiedMap
  let $code := $q/@code
```

```
 let $riverIds := $q/riverId/text()
 for $river in //river
 where $river/@id = $riverIds
  and (
    some $loc in $river/located satisfies $loc/@country = $code
  )
 return
  <result>
    <country>{ $code }</country>
    <river>{ $river/name[1] }</river>
  </result>
```

XQuery 3: Të listohen të gjitha detet në të cilat nuk ka asnjë ishull mirëpo shtetet të cilat kufizohet ai det janë anëtare në NATO ose BE.

```
(: Get all seas that have no islands :)
for $sea in //sea
where not(//island[@sea = $sea/@id])

(: Get the list of country codes that border this sea :)
let $countries := tokenize($sea/@country, '\s+')

(: Filter only countries that are members of NATO or EU :)
let $validCountries :=
  for $code in $countries
  let $country := //country[@car_code = $code]
  where contains($country/@memberships, "org-NATO")
    or contains($country/@memberships, "org-EU")
  return $code

(: Ensure that all countries bordering the sea are in NATO or EU :)
where count($validCountries) = count($countries)

(: Return the result showing the sea name :)
return
  <result>
    <sea>{ $sea/name }</sea>
  </result>
```

XQuery 4: Të listohen të gjitha shtetet anëtare të NATO-s të cilat kanë dalje në të njëjtin det dhe liqen.

```
declare function local:ordered-pair($a as xs:string, $b as xs:string) as xs:string {
  if ($a le $b) then concat($a, '|', $b)
  else concat($b, '|', $a)
};

let $natoCodes := distinct-values(
  for $c in //country
  where contains($c/@memberships, "org-NATO")
  return $c/@car_code
)

(: Create set of pairs for seas :)
let $seaPairs := distinct-values(
  for $sea in //sea
  let $codes := tokenize($sea/@country, '\s+')
  let $nato := for $c in $codes where $c = $natoCodes return $c
  where count($nato) > 1
  for $i in 1 to count($nato) - 1
  for $j in $i + 1 to count($nato)
  return local:ordered-pair($nato[$i], $nato[$j])
)

(: Create set of pairs for lakes :)
let $lakePairs := distinct-values(
  for $lake in //lake
  let $codes := tokenize($lake/@country, '\s+')
  let $nato := for $c in $codes where $c = $natoCodes return $c
  where count($nato) > 1
  for $i in 1 to count($nato) - 1
  for $j in $i + 1 to count($nato)
  return local:ordered-pair($nato[$i], $nato[$j])
)

(: Only keep pairs that appear in both sets (sea + lake) :)
let $strictMatches := distinct-values(
  for $p in $seaPairs
  where $p = $lakePairs
  return $p
)

return
  for $pair in $strictMatches
  let $codes := tokenize($pair, '\|')
  let $name1 := //country[@car_code = $codes[1]]/name[1]
```

```
  let $name2 := //country[@car_code = $codes[2]]/name[1]
  return
    <pair>
      <country1>{ $name1 }</country1>
      <country2>{ $name2 }</country2>
    </pair>
```

XQuery 5: Të listohen 5 malet më të larta të cilat shtrihen në vendet e Ballkanit duke përjashtuar Serbinë.

```
let $balkanCodes := ("AL", "GR", "MK", "BG", "BA", "HR", "ME", "XK", "RO", "SI")

let $balkanMountains :=
  for $mountain in //mountain
  let $codes := tokenize($mountain/@country, '\s+')
  where some $c in $codes satisfies $c = $balkanCodes
  order by number($mountain/elevation) descending
  return $mountain

return
  for $m in $balkanMountains[position() <= 5]
  return
    <mountain>
      <name>{ $m/name/text() }</name>
      <elevation>{ $m/elevation/text() }</elevation>
      <country>{ $m/@country }</country>
    </mountain>
```

XPath 1: Listo emrat e shteteve të cilat janë anëtare të NATO-s dhe kanë popullsi më pak se 10000000 banorë të regjistruar në vitin 2011.

```
//country[
  contains(@memberships, 'org-NATO') and
  population[@year="2011"] < 10000000
]/name
```

XPath 2: Listo emrat e kryeqyteteve të shteteve të cilat janë anëtare të EU-së dhe nëpër këto kryeqytete nuk kalon asnjë lum.

```
//city[
  @id = //country[contains(@memberships, 'org-EU')]/@capital
  and not(located_at[@watertype = 'river'])
]/name[1]/text()
```

XPath 3: Listo të gjitha shkretëtirat të cilat gjendën në Etiopi dhe janë më të mëdha se 150000m^2.

```
//desert[contains(@country, 'ETH') and number(area) > 150000]/name
```

XPath 4: Listo të gjitha aeroportet në perëndim të Shteteteve të Bashkuara dhe kanë lartësi mbidetare më të vogël se 50m.

```
//airport[
  @country = 'USA' and
  number(longitude) < -120 and
  number(elevation) < 50
]/name/text()
```

XPath5: Listo emrat e vendeve në Evropë që kanë pasur një popullësi më të madhe se 10M banorë në vitin 2011.

```
//country[
  encompassed/@continent = 'europe' and
  number(population[@year = '2011']/text()) > 10000000
]/name/text()
```

Measure për të pasur një titull dinamik në vizualizimin për temperaturat vjetore në qytetet kryesore:

```
TemperatureTitle = "Temperature Trend for " & SELECTEDVALUE('city_temperature'[City],
"Selected City")
```

Measure që llogaritë numrin maksimal të liqeneve tek vizualizimi i numrit të liqeneve për shtet:

```
MaxLakeCount = MAX('Table'[LakeCount])
```

Measure që llogaritë numrin minimal të liqeneve tek vizualizimi i numrit të liqeneve për shtet:

```
MinLakeCount = MIN('Table'[LakeCount])
```

Measure që gjen shtetin me numrin maksimal të liqeneve tek vizualizimi i numrit të liqeneve për shtet:

```
MaxLakeCountry =
CALCULATE(
    SELECTEDVALUE('Table'[Country]),
    FILTER('Table', 'Table'[LakeCount] = [MaxLakeCount])
)
```

Measure që gjen shtetin me numrin minimal të liqeneve tek vizualizimi i numrit të liqeneve për shtet:

```
MinLakeCountry =
CALCULATE(
    SELECTEDVALUE('Table'[Country]),
    FILTER('Table', 'Table'[LakeCount] = [MinLakeCount])
)
```

View e cila liston lumenjtë të cilët rrjedhin nëpër shtete të cilat nuk janë anëtarë të NATO-s, kanë dalje në dete si dhe numri i luemnjëve në këto shtete është më i madh se 10, kjo view përdoret në vizualizimin e lumenjëve sipas shteteve:

```
 CREATE VIEW View2
AS
SELECT DISTINCT gr.country, gr.river
FROM geo_river gr
WHERE gr.country IN (
   -- shtetet qe kane dalje ne deti
   SELECT DISTINCT c.Code
   FROM country c
   JOIN geo_sea gs ON c.Code = gs.Country
)
AND gr.country NOT IN (
   -- shtetet qe jane anetare te NATO
   SELECT im.Country
   FROM isMember im
   WHERE im.Organization = 'NATO' AND im.Type = 'member'
)
AND gr.country IN (
   -- shtetet me 10 e me shume lumenj
   SELECT country
   FROM geo_river
   GROUP BY country
   HAVING COUNT(DISTINCT river) > 10
);
```