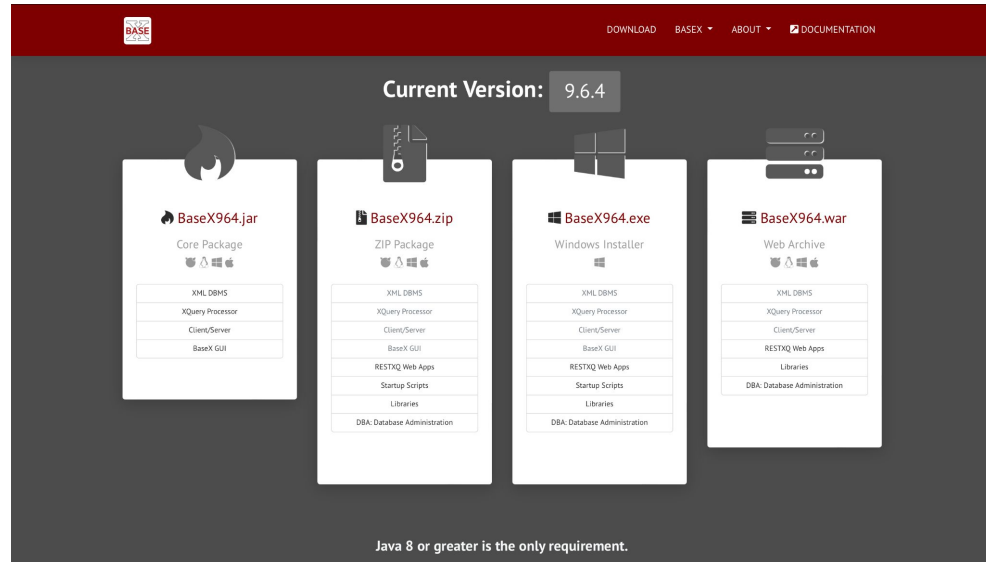




# Gestión de BBDD XML con BaseX

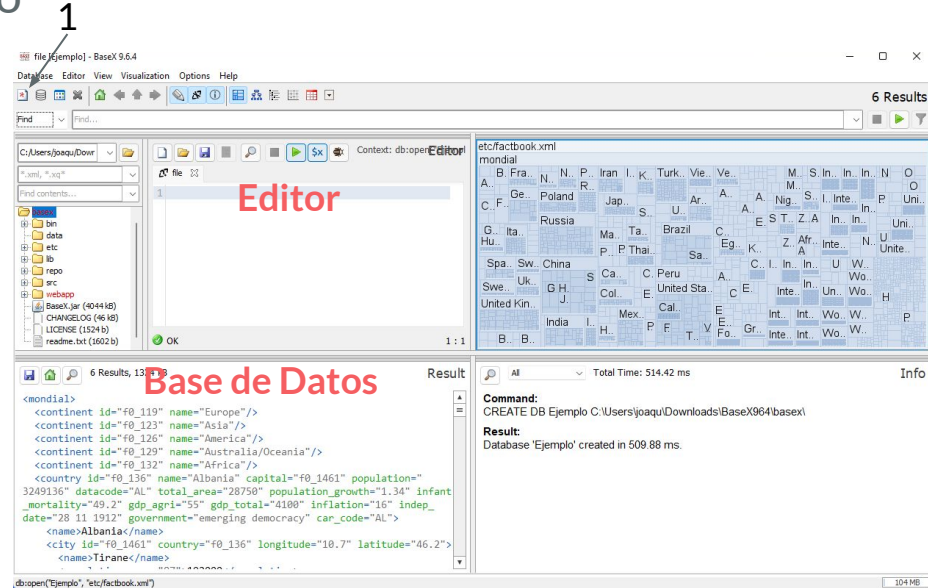
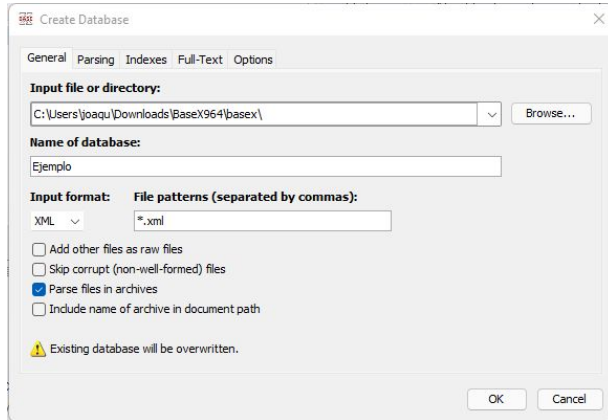
# Instalación de BaseX

1. Descargar Basex964.zip desde [basex.org/download/](https://basex.org/download/)
2. Descomprimir el fichero
3. Ejecutar BaseX.jar



# Entorno de trabajo

1. Creamos una nueva BBDD
2. Le damos nombre a la base de datos
3. Se crea una base de datos de ejemplo



# Consultas XPath

Las consultas en XPath siguen la siguiente estructura:

`//Etiqueta` : Obtenemos la etiquetas con ese nombre con todo su contenido

`//Etiqueta/@atributo` : Obtenemos el atributo y su valor de cada etiqueta

`//Etiqueta/@atributo/string()` : Obtenemos el valor del atributo buscado

# Consultas XPath

//country: Obtenemos toda la información de los países

The screenshot displays the BaseX 8.6.4 XML editor interface. The top menu bar includes File, Edit, View, Visualization, Options, and Help. The main window is divided into three panes: a left pane showing the file explorer with a project named 'C:\Users\joequ\Downloads\BasesX96\ba', a central editor pane showing the XPath query `//country`, and a right pane showing the results of the query.

The right pane displays 231 results, which are XML elements representing countries. The results are shown in a grid-like format, with each row representing a country and its associated data. The results are sorted by the 'country' element, and the 'Total Time' for the query is 49.34 ms.

The bottom pane shows the query results in a structured format, including the query itself, the result set, and the timing information. The query is `(db open-pre('Ejemplo', 0), ...)descendant: country`, and the result set contains 231 items. The timing information shows that the query was executed in 49.34 ms, with a total time of 49.34 ms.

**Compiling:**

- rewrite context value to document-node() sequence: -> (db open-pre('Ejemplo', 0), ...)
- rewrite uri root(nodes) to document-node() sequence: uri root((db open-pre('Ejemplo', 0), ...)) -> (db open-pre('Ejemplo', 0), ...)
- merge: descendant: country

**Optimized Query:**

(db open-pre('Ejemplo', 0), ...)descendant: country

**Query:**

//country

**Result:**

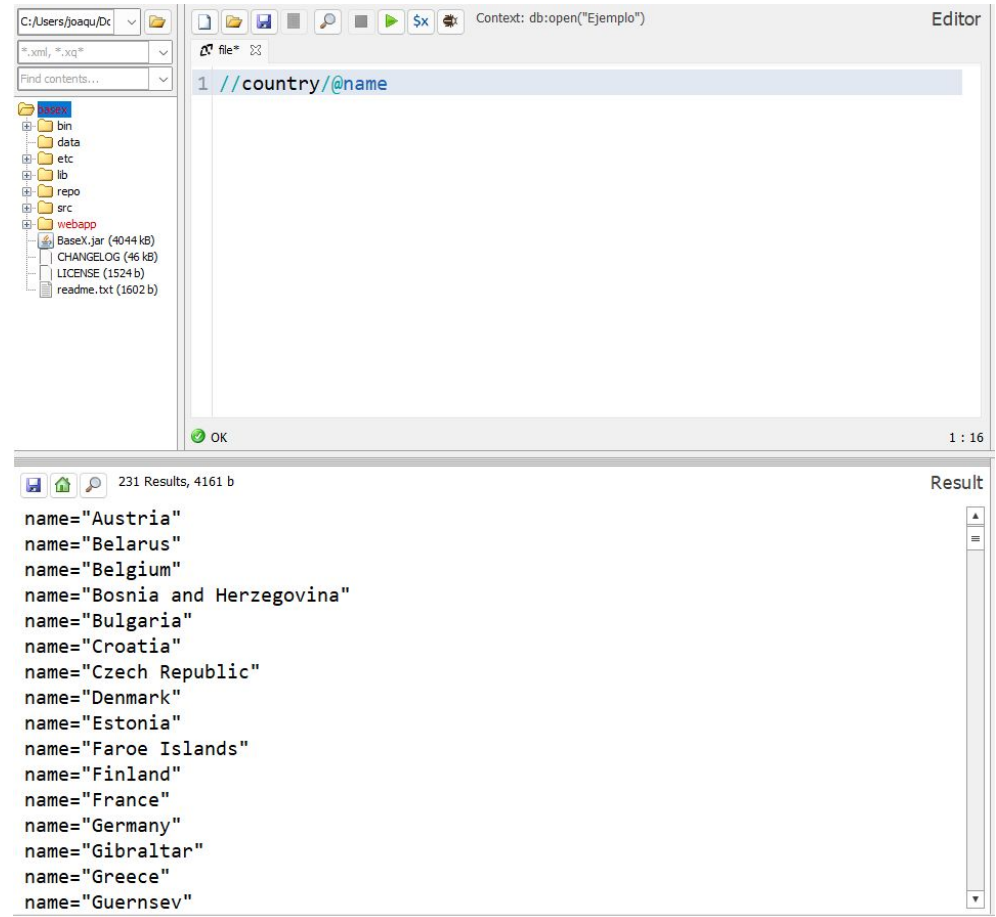
- Hits: 231 items
- Updated: 0 items
- Printed: 831 kB
- Read Locking: Ejemplo
- Write Locking: (none)

**Timing:**

- Parsing: 0.22 ms
- Compiling: 9.0 ms
- Evaluating: 18.11 ms
- Printing: 22.2 ms
- Total Time: 49.54 ms

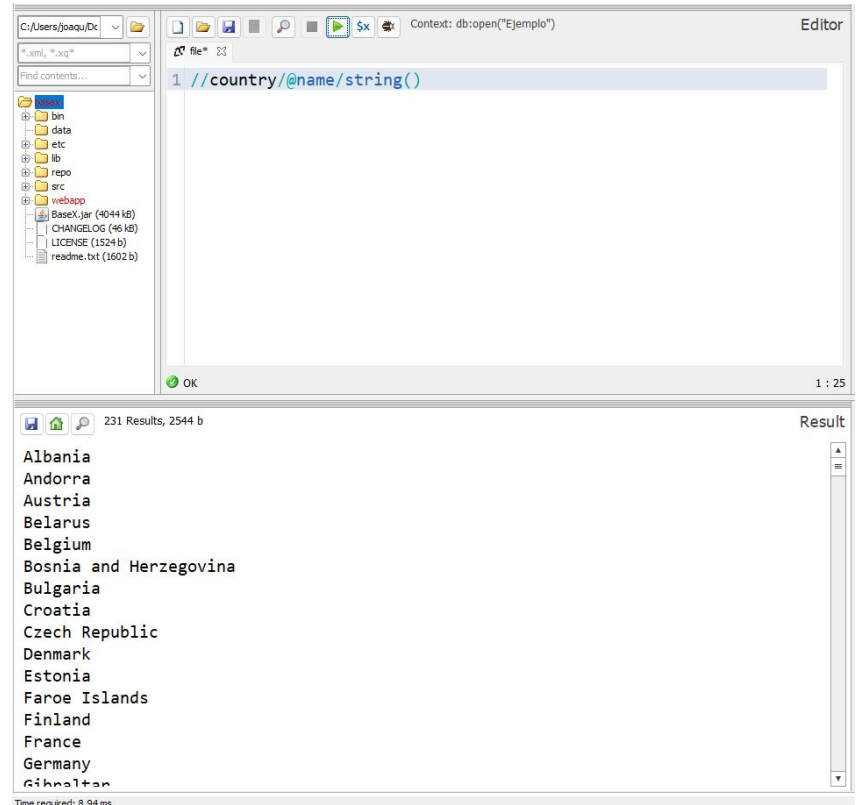
# Consultas XPath

`//country/@name` : Obtenemos el atributo name con el nombre de cada país



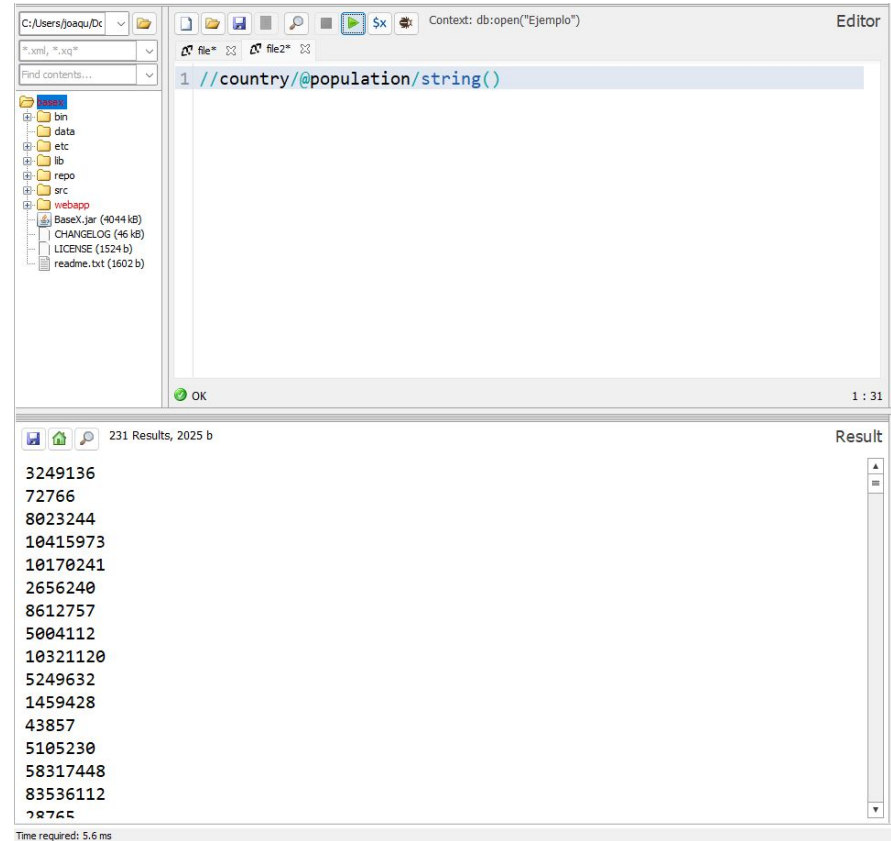
# Consultas XPath

`//country/@name/string()` : Obtenemos los nombres de los países de la base de datos como un string



# Consultas XPath

`//country/@population/string()` :  
Obtenemos la población de los países de  
la base de datos como un string

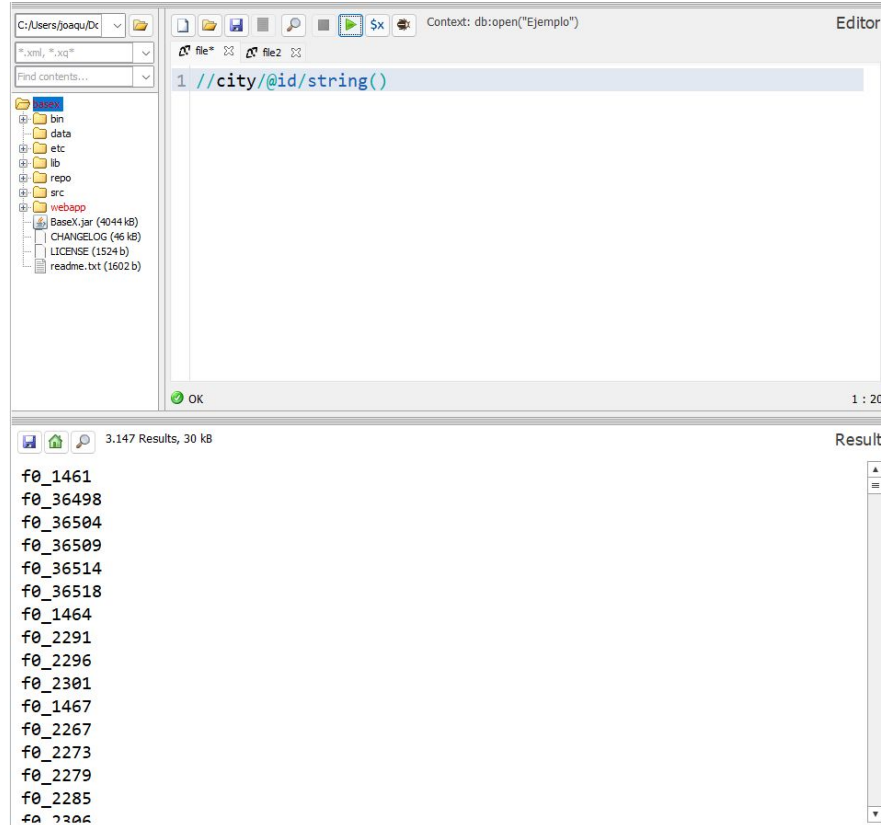




# Consultas XPath

**Ejercicio:** ¿Cómo encontrarías ahora el id de las ciudades de la base de datos?

# Consultas XPath



# Consultas XQuery (Flowr)

Se le llama flowr por la estructura de sus consultas, ya que es for-let-order-where-return:

```
for $pais in //country
```

```
let $poblacion := xs:integer($pais/@population)
```

```
order by $pais/@population/number()
```

```
where substring($pais/@name/string(), 1, 1) = "A"
```

```
return
```

```
<pais nombre="{ $pais/name/string()}" habitantes="{ $poblacion}" />
```

# Consultas XQuery (Flowr)

- Usamos \$var para crear variables.
- := para asignaciones
- <> para el texto que queremos mostrar como salida.
  - “{}” para mostrar el contenido de una variable

for \$continente in //continent

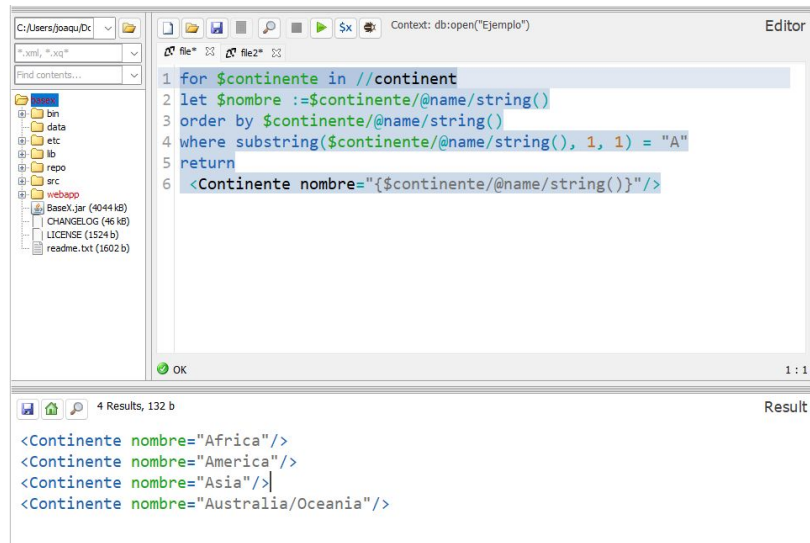
let \$nombre := \$continente/@name/string()

order by \$continente/@name/string()

where substring(\$continente/@name/string(), 1, 1) = "A"

return

<Continente nombre="{ \$continente/@name/string() }"/>



# Consultas XQuery (Flowr)

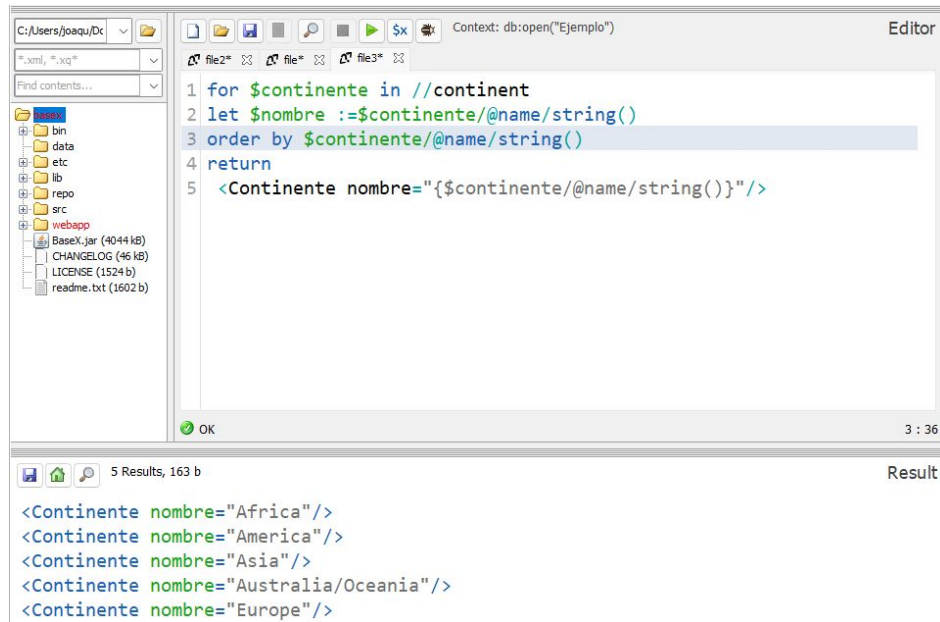
for \$continente in //continent

let \$nombre := \$continente/@name/string()

order by \$continente/@name/string()

return

<Continente nombre="{ \$continente/@name/string()}"/>



# Consultas XQuery (Flowr)

for \$pais in //country

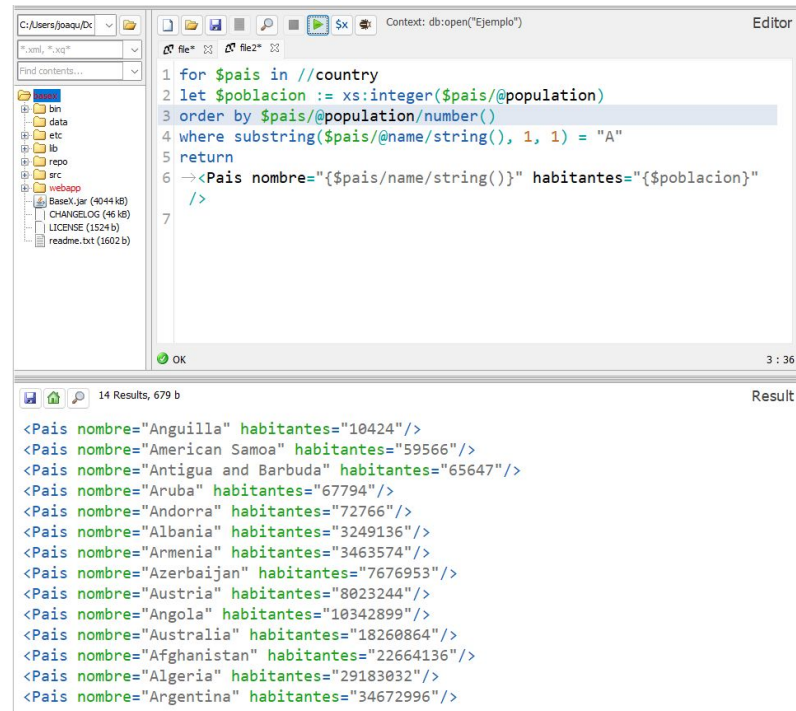
let \$poblacion := xs:integer(\$pais/@population)

order by \$pais/@population/number()

where substring(\$pais/@name/string(), 1, 1) = "A"

return

<pais nombre="{ \$pais/name/string()}" habitantes="{ \$poblacion}" />



The screenshot shows the Flowr XQuery editor interface. The top pane displays the XQuery code for querying a database of countries, filtered by those starting with 'A' and ordered by population. The bottom pane shows the resulting XML output, which is a list of country elements with their names and populations.

```
1 for $pais in //country
2 let $poblacion := xs:integer($pais/@population)
3 order by $pais/@population/number()
4 where substring($pais/@name/string(), 1, 1) = "A"
5 return
6   <Pais nombre="{ $pais/name/string()}" habitantes="{ $poblacion}"
7   />
```

14 Results, 679 b

```
<Pais nombre="Anguilla" habitantes="10424"/>
<Pais nombre="American Samoa" habitantes="59566"/>
<Pais nombre="Antigua and Barbuda" habitantes="65647"/>
<Pais nombre="Aruba" habitantes="67794"/>
<Pais nombre="Andorra" habitantes="72766"/>
<Pais nombre="Albania" habitantes="3249136"/>
<Pais nombre="Armenia" habitantes="3463574"/>
<Pais nombre="Azerbaijan" habitantes="7676953"/>
<Pais nombre="Austria" habitantes="8023244"/>
<Pais nombre="Angola" habitantes="10342899"/>
<Pais nombre="Australia" habitantes="18260864"/>
<Pais nombre="Afghanistan" habitantes="22664136"/>
<Pais nombre="Algeria" habitantes="29183032"/>
<Pais nombre="Argentina" habitantes="34672996"/>
```

# Consultas XQuery

**Ejercicio:** Cambia la consulta para que aparezca  
España

# Consultas XQuery

Ejemplo

The screenshot shows a software interface for editing and executing XQuery. The top-left pane displays a file explorer with a tree view containing folders like 'bin', 'data', 'etc', 'lib', 'repo', 'src', and 'webapp', along with files like 'BaseX.jar', 'CHANGELOG', 'LICENSE', and 'readme.txt'. The top-right pane is the 'Editor' window, showing an XQuery script with line numbers 1 through 7. The script filters for countries with 'Spa' in their names and orders them by population. The bottom pane is the 'Result' window, showing a single XML result element for Spain with its population.

Context: db:open("Ejemplo")

```
1 for $pais in //country
2 let $poblacion := xs:integer($pais/@population)
3 order by $pais/@population/number()
4 where substring($pais/@name/string(), 1, 3) = "Spa"
5 return
6 →<Pais nombre="{ $pais/name/string()}" habitantes="{ $poblacion}"
7 />
```

OK 4 : 51

1 Result, 44 b

```
<Pais nombre="Spain" habitantes="39181112"/>
```

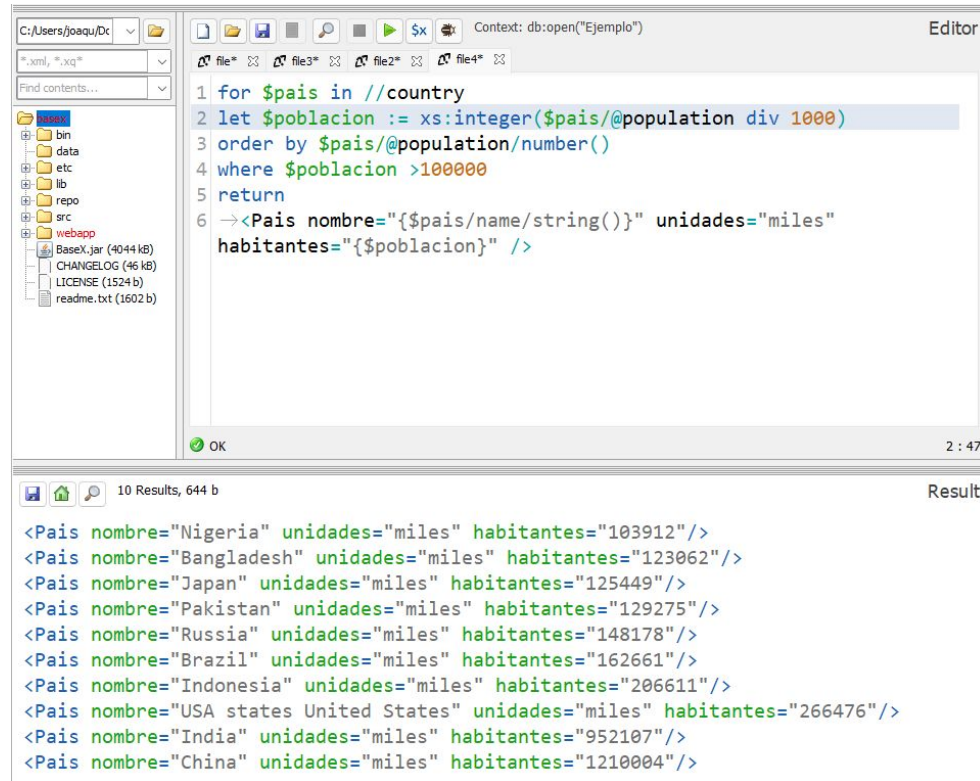
rezca



# Consultas XQuery

**Ejercicio:** Cambia la consulta para que aparezcan los países con población mayor de 100000 millones.

# Consultas XQuery



The screenshot displays an XQuery editor and its results. The editor window, titled "Editor", shows the following XQuery code:

```
1 for $pais in //country
2 let $poblacion := xs:integer($pais/@population div 1000)
3 order by $pais/@population/number()
4 where $poblacion >100000
5 return
6 -><Pais nombre="{ $pais/name/string()}" unidades="miles"
   habitantes="{ $poblacion}" />
```

The left sidebar shows a file explorer with a tree view containing folders like "bin", "data", "etc", "lib", "repo", "src", and "webapp", along with files like "BaseX.jar", "CHANGELOG", "LICENSE", and "readme.txt".

The bottom section, titled "Result", shows the output of the query, which is 10 XML results (644 bytes). The results are as follows:

```
<Pais nombre="Nigeria" unidades="miles" habitantes="103912"/>
<Pais nombre="Bangladesh" unidades="miles" habitantes="123062"/>
<Pais nombre="Japan" unidades="miles" habitantes="125449"/>
<Pais nombre="Pakistan" unidades="miles" habitantes="129275"/>
<Pais nombre="Russia" unidades="miles" habitantes="148178"/>
<Pais nombre="Brazil" unidades="miles" habitantes="162661"/>
<Pais nombre="Indonesia" unidades="miles" habitantes="206611"/>
<Pais nombre="USA states United States" unidades="miles" habitantes="266476"/>
<Pais nombre="India" unidades="miles" habitantes="952107"/>
<Pais nombre="China" unidades="miles" habitantes="1210004"/>
```

# Funciones XQuery

En XQuery podemos crear funciones locales para no repetir el código, para ello se crean de la siguiente manera:

```
declare function local:miles($valor as xs:decimal?)  
as xs:decimal?  
{  
  ///Contenido función  
};
```

# Funciones XQuery

The screenshot shows an XQuery editor window with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like BaseX.jar, CHANGELOG, LICENSE, and readme.txt. The code editor contains an XQuery script that defines a local function to calculate population in miles and then queries a database for countries with a population greater than 100,000. The result viewer at the bottom shows 10 results, each representing a country with its name, population in miles, and total population.

Context: db:open("Ejemplo")

```
1 declare function local:miles($valor as xs:decimal?)
2 as xs:decimal?
3 {
4   let $salida := xs:integer ($valor div 1000)
5   return $salida
6 };
7
8
9 for $pais in //country
10 let $poblacion := local:miles($pais/@population)
11 order by $pais/@population/number()
12 where $poblacion >100000
13 return
14 -><Pais nombre="{ $pais/name/string()}" unidades="miles"
    habitantes="{ $poblacion}" />
```

10 : 38

10 Results, 644 b

```
<Pais nombre="Nigeria" unidades="miles" habitantes="103912"/>
<Pais nombre="Bangladesh" unidades="miles" habitantes="123062"/>
<Pais nombre="Japan" unidades="miles" habitantes="125449"/>
<Pais nombre="Pakistan" unidades="miles" habitantes="129275"/>
<Pais nombre="Russia" unidades="miles" habitantes="148178"/>
<Pais nombre="Brazil" unidades="miles" habitantes="162661"/>
<Pais nombre="Indonesia" unidades="miles" habitantes="206611"/>
<Pais nombre="USA states United States" unidades="miles" habitantes="266476"/>
<Pais nombre="India" unidades="miles" habitantes="952107"/>
<Pais nombre="China" unidades="miles" habitantes="1210004"/>
```

# Ordenar la salida en XQuery

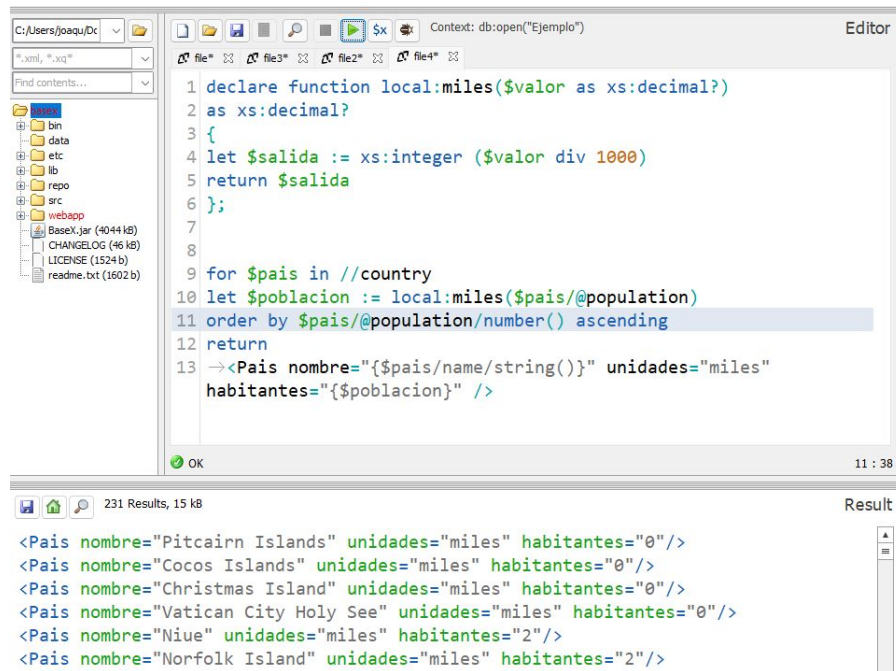
Podemos ordenar los datos de forma ascendente o descendente, para ello usamos las cláusulas:

- Ascending: Ordena los datos de manera ascendente
- Descending: Ordena los datos de manera descendente

# Ordenar la salida en XQuery

Podemos ordenar los datos de forma ascendente o descendente, para ello usamos las cláusulas:

- Ascending: Ordena los datos de manera ascendente



The screenshot shows an XQuery editor with a file explorer on the left and a code editor on the right. The code editor contains an XQuery script that defines a local function `local:miles` and uses it to query a database for country names and populations, ordered by population in ascending order. The results window at the bottom displays the output as XML elements.

```
1 declare function local:miles($valor as xs:decimal?)
2 as xs:decimal?
3 {
4 let $salida := xs:integer ($valor div 1000)
5 return $salida
6 };
7
8
9 for $pais in //country
10 let $poblacion := local:miles($pais/@population)
11 order by $pais/@population/number() ascending
12 return
13 -><Pais nombre="{ $pais/name/string()}" unidades="miles"
    habitantes="{ $poblacion}" />
```

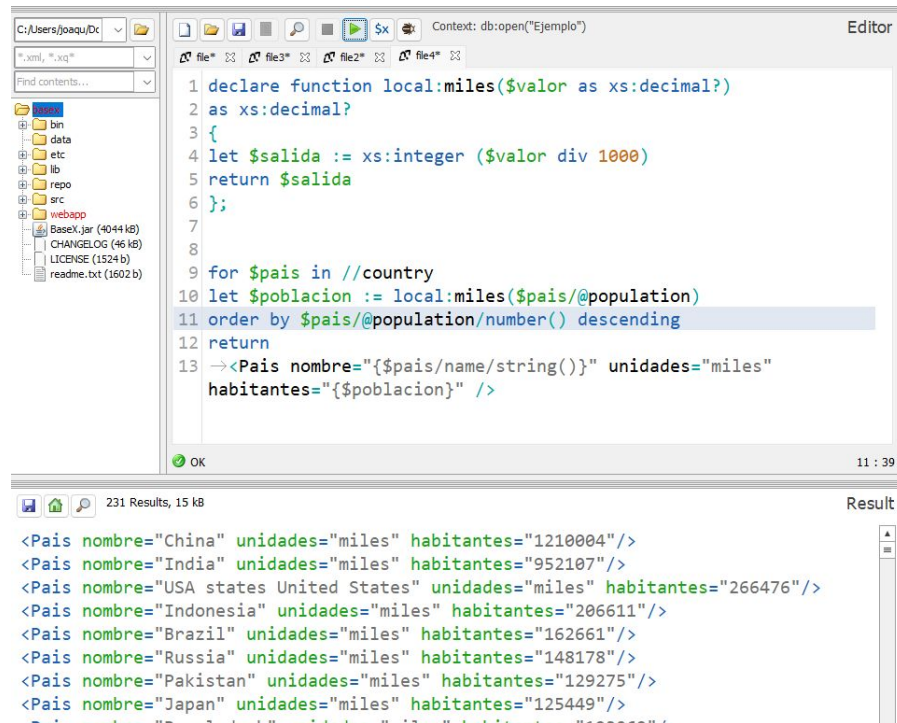
231 Results, 15 kB

```
<Pais nombre="Pitcairn Islands" unidades="miles" habitantes="0"/>
<Pais nombre="Cocos Islands" unidades="miles" habitantes="0"/>
<Pais nombre="Christmas Island" unidades="miles" habitantes="0"/>
<Pais nombre="Vatican City Holy See" unidades="miles" habitantes="0"/>
<Pais nombre="Niue" unidades="miles" habitantes="2"/>
<Pais nombre="Norfolk Island" unidades="miles" habitantes="2"/>
```

# Ordenar la salida en XQuery

Podemos ordenar los datos de forma ascendente o descendente, para ello usamos las cláusulas:

- Descending: Ordena los datos de manera descendente



The screenshot shows an XQuery editor window with a file explorer on the left and a code editor on the right. The code defines a function `local:miles` that takes a decimal value and returns its integer part divided by 1000. It then queries a database for country data, ordered by population in descending order. The results window at the bottom shows the output as XML, listing countries like China, India, and USA with their population in miles.

```
1 declare function local:miles($valor as xs:decimal?)
2 as xs:decimal?
3 {
4   let $salida := xs:integer ($valor div 1000)
5   return $salida
6 };
7
8
9 for $pais in //country
10 let $poblacion := local:miles($pais/@population)
11 order by $pais/@population/number() descending
12 return
13 <Pais nombre="{ $pais/name/string()}" unidades="miles"
    habitantes="{ $poblacion}" />
```

231 Results, 15 kB

```
<Pais nombre="China" unidades="miles" habitantes="1210004"/>
<Pais nombre="India" unidades="miles" habitantes="952107"/>
<Pais nombre="USA states United States" unidades="miles" habitantes="266476"/>
<Pais nombre="Indonesia" unidades="miles" habitantes="206611"/>
<Pais nombre="Brazil" unidades="miles" habitantes="162661"/>
<Pais nombre="Russia" unidades="miles" habitantes="148178"/>
<Pais nombre="Pakistan" unidades="miles" habitantes="129275"/>
<Pais nombre="Japan" unidades="miles" habitantes="125449"/>
```

# Comentarios en XQuery

Para comentar una parte del código tenemos que poner este entre (: Comentario :)

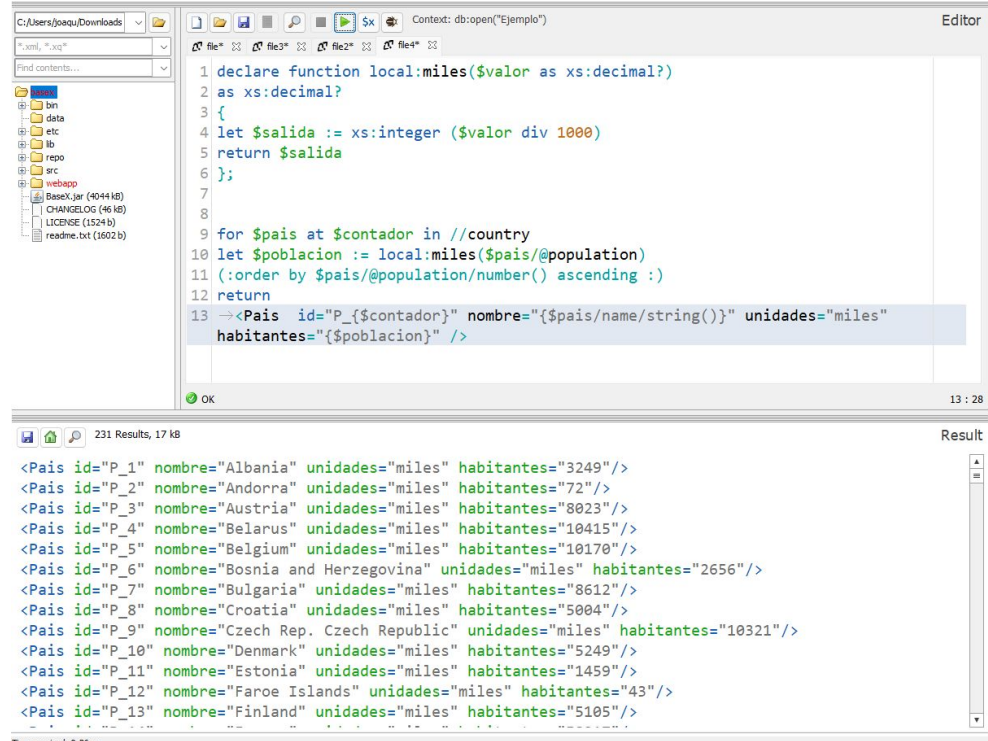
```
9 for $pais in //country
10 let $poblacion := local:miles($pais/@population)
11 (:order by $pais/@population/number() ascending :)
12 return
13 →<Pais nombre="{ $pais/name/string()}" unidades="miles" habitantes="{ $poblacion}"
   />
```



# Contadores en for XQuery

Podemos usar un contador de incremento automático para cada ocurrencia de una etiqueta

for \$pais at \$contador in //country



```
1 declare function local:miles($valor as xs:decimal?)
2 as xs:decimal?
3 {
4   let $salida := xs:integer ($valor div 1000)
5   return $salida
6 };
7
8
9 for $pais at $contador in //country
10 let $poblacion := local:miles($pais/@population)
11 (:order by $pais/@population/number() ascending :)
12 return
13 -><Pais id="P_{ $contador}" nombre="{ $pais/name/string()}" unidades="miles"
    habitantes="{ $poblacion}" />
```

231 Results, 17 kB

```
<Pais id="P_1" nombre="Albania" unidades="miles" habitantes="3249"/>
<Pais id="P_2" nombre="Andorra" unidades="miles" habitantes="72"/>
<Pais id="P_3" nombre="Austria" unidades="miles" habitantes="8023"/>
<Pais id="P_4" nombre="Belarus" unidades="miles" habitantes="10415"/>
<Pais id="P_5" nombre="Belgium" unidades="miles" habitantes="10170"/>
<Pais id="P_6" nombre="Bosnia and Herzegovina" unidades="miles" habitantes="2656"/>
<Pais id="P_7" nombre="Bulgaria" unidades="miles" habitantes="8612"/>
<Pais id="P_8" nombre="Croatia" unidades="miles" habitantes="5004"/>
<Pais id="P_9" nombre="Czech Rep. Czech Republic" unidades="miles" habitantes="10321"/>
<Pais id="P_10" nombre="Denmark" unidades="miles" habitantes="5249"/>
<Pais id="P_11" nombre="Estonia" unidades="miles" habitantes="1459"/>
<Pais id="P_12" nombre="Faroe Islands" unidades="miles" habitantes="43"/>
<Pais id="P_13" nombre="Finland" unidades="miles" habitantes="5105"/>
```

# Contadores en for XQuery

**Ejercicio:** ¿Qué id tienen los países con entre 3.000 y 10.000 habitantes?

# Contadores en for XQuery

The screenshot displays an XQuery editor interface. On the left is a file explorer showing a directory structure with folders like 'bin', 'data', 'etc', 'lib', 'repo', 'src', and 'webapp', along with files like 'BaseX.jar', 'CHANGELOG', 'LICENSE', and 'readme.txt'. The main editor area contains the following XQuery code:

```
1 declare function local:miles($valor as xs:decimal?)
2 as xs:decimal?
3 {
4   let $salida := xs:integer ($valor div 1000)
5   return $salida
6 };
7
8
9 for $pais at $contador in //country
10 let $poblacion := local:miles($pais/@population)
11 (:order by $pais/@population/number() ascending :)
12 where $poblacion < 10
13 where $poblacion > 3
14 return
15 →<Pais id="P_{$contador}" nombre="{ $pais/name/string()}" unidades="miles"
    habitantes="{ $poblacion}" />
```

Below the editor, a status bar shows a green checkmark and 'OK'. The bottom section, labeled 'Result', shows the output of the query:

2 Results, 159 b

```
<Pais id="P_137" nombre="Saint Pierre and Miquelon" unidades="miles" habitantes="6"/>
<Pais id="P_171" nombre="Saint Helena" unidades="miles" habitantes="6"/>
```

# Consultas XQuery

Vamos ahora a hacer una consulta que nos muestre las ciudades con más de 1 millón de habitantes y en el momento de la última medición.

En esta consulta usamos `//etiqueta/etiquetaHija` para acceder a una etiqueta descendiente

The screenshot shows an XQuery editor interface. The left pane displays a file explorer for the path `C:/Users/joaqu/Downloads`, showing various files and folders. The main editor pane contains the following XQuery code:

```
1 for $ciudad in //city
2 let $nombre := $ciudad/name/string()
3 let $poblacion := $ciudad/population[last()]
4 order by $poblacion/number() descending
5 where $poblacion/number() > 1000000
6 return
7 <dato>{$nombre} en el año {$poblacion/@year/data()} tenía {$poblacion/data()}
   habitantes</dato>
```

The bottom pane shows the results of the query, displaying 236 results in 15 kB. The results are listed as XML fragments:

```
<dato>Seoul en el año 95 tenía 10229262 habitantes</dato>
<dato>Mumbai Greater en el año 91 tenía 9925891 habitantes</dato>
<dato>Karachi en el año 95 tenía 9863000 habitantes</dato>
<dato>Mexico Mexico City Ciudad de Mexico en el año 90 tenía 9815795 habitantes</dato>
<dato>Sao Paulo en el año 96 tenía 9811776 habitantes</dato>
```

# Consultas XQuery

**Ejercicio:** Añade el identificador de las ciudades  
que encontramos

# Consultas XQuery

Ejerc

ades

The screenshot shows the BaseX XQuery editor interface. The left sidebar displays a file explorer with the following contents:

- baseX
- bin
- data
- etc
- lib
- repo
- src
- webapp
- BaseX.jar (4044 kB)
- CHANGELOG (46 kB)
- LICENSE (1524 b)
- readme.txt (1602 b)

The main editor window displays the following XQuery:

```
1 for $ciudad at $contador in //city
2 let $nombre := $ciudad/name/string()
3 let $poblacion := $ciudad/population[last()]
4 order by $poblacion/number() descending
5 where $poblacion/number() > 1000000
6 return
7 <dato>{$nombre} con id {$contador} en el año {$poblacion/@year/data()} tenía {$poblacion/data()} habitantes</dato>
```

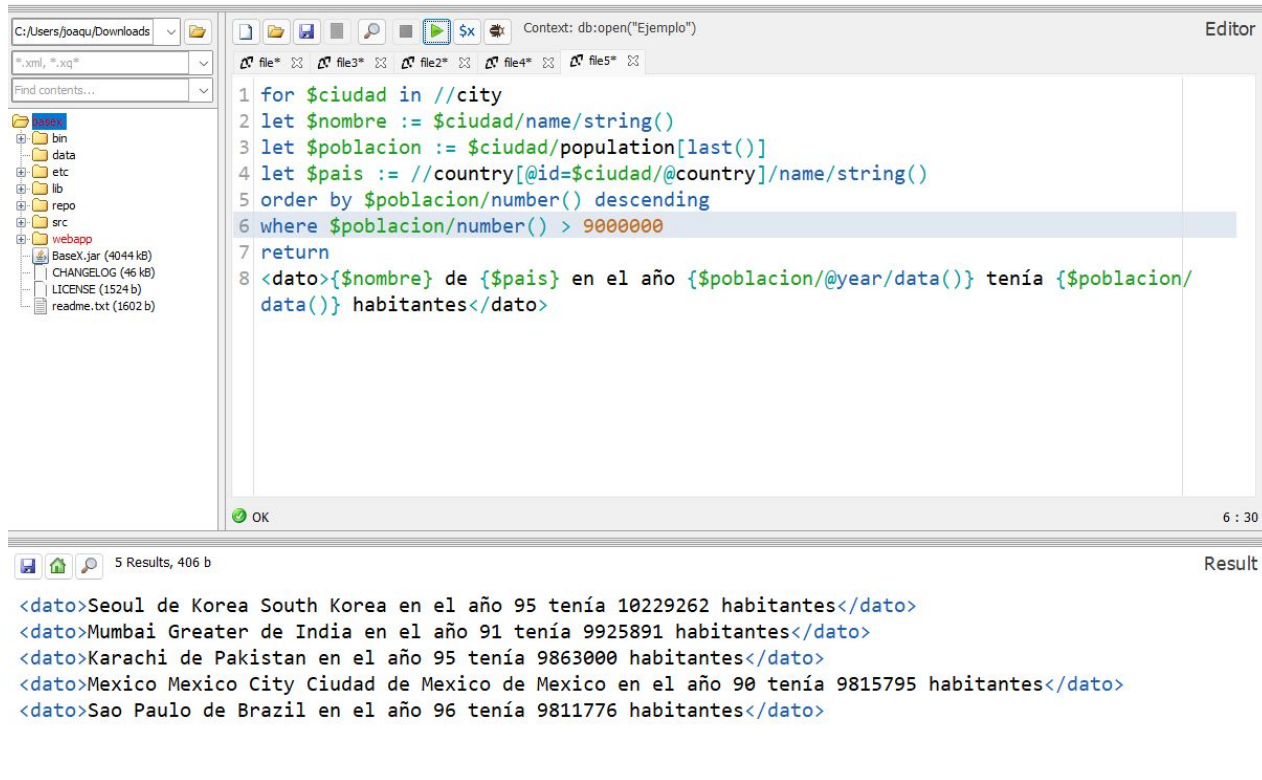
The status bar at the bottom indicates "236 Results, 18 kB". The results pane on the right shows the following output:

```
<dato>Seoul con id 1638 en el año 95 tenía 10229262 habitantes</dato>
<dato>Mumbai Greater con id 1365 en el año 91 tenía 9925891 habitantes</dato>
<dato>Karachi con id 1691 en el año 95 tenía 9863000 habitantes</dato>
<dato>Mexico Mexico City Ciudad de Mexico con id 2413 en el año 90 tenía 9815795 habitantes</dato>
<dato>Sao Paulo con id 2104 en el año 96 tenía 9811776 habitantes</dato>
<dato>Moscow con id 489 en el año 95 tenía 8717000 habitantes</dato>
```

# Consultas XQuery

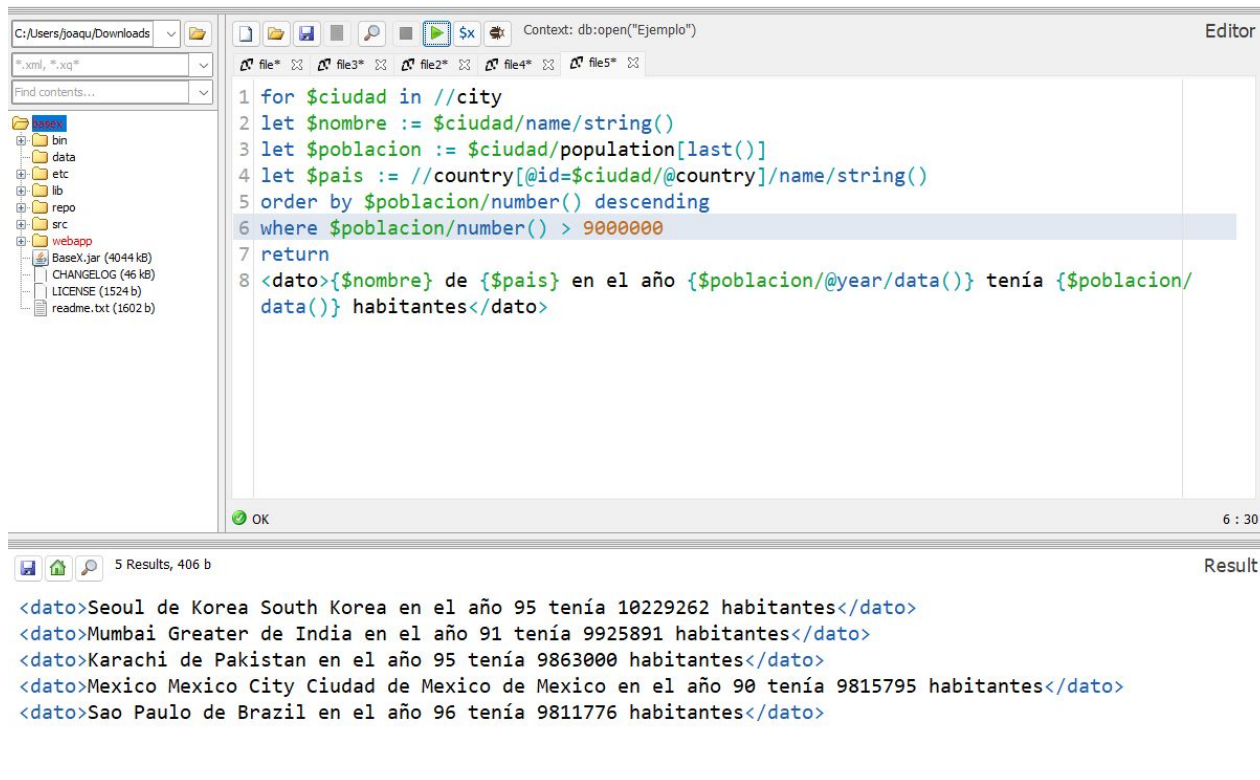
Vamos ahora a hacer una consulta que nos muestre las ciudades con más de 1 millón de habitantes y el país al que pertenecen en el momento de la última medición.

Recordamos que para acceder a una etiqueta concreta podemos acceder a ella por medio de `//etiqueta[valor]`



# Consultas XQuery

Ahora vamos a corregir los errores, ya que salen dos nombres de países en alguno de los resultados.



The screenshot shows a database editor interface. The left pane displays a file explorer for the path `C:/Users/joaqu/Downloads`, showing a folder named `base` and several files including `BaseX.jar`, `CHANGELOG`, `LICENSE`, and `readme.txt`. The main editor pane shows an XQuery script with the following lines:

```
1 for $ciudad in //city
2 let $nombre := $ciudad/name/string()
3 let $poblacion := $ciudad/population[last()]
4 let $pais := //country[@id=$ciudad/@country]/name/string()
5 order by $poblacion/number() descending
6 where $poblacion/number() > 9000000
7 return
8 <dato>{$nombre} de {$pais} en el año {$poblacion/@year/data()} tenía {$poblacion/
data()} habitantes</dato>
```

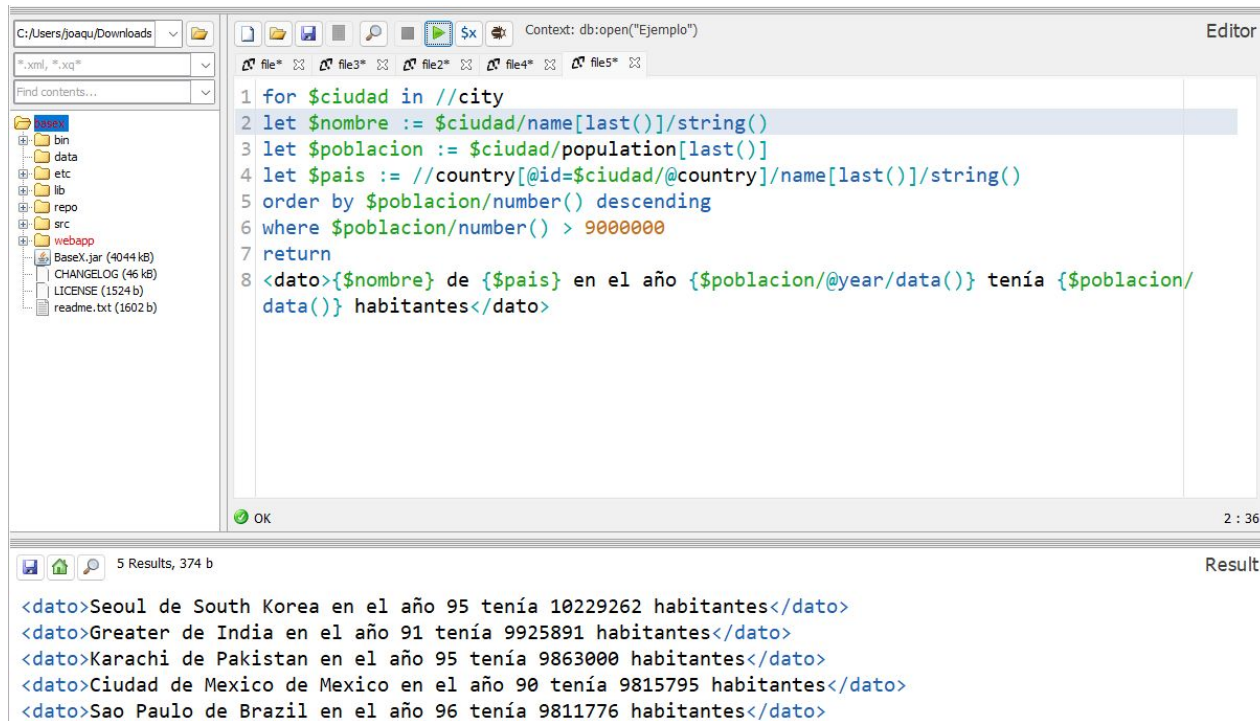
The script is executed, and the bottom pane shows the results. The status bar indicates "5 Results, 406 b". The results are displayed as XML fragments:

```
<dato>Seoul de Korea South Korea en el año 95 tenía 10229262 habitantes</dato>
<dato>Mumbai Greater de India en el año 91 tenía 9925891 habitantes</dato>
<dato>Karachi de Pakistan en el año 95 tenía 9863000 habitantes</dato>
<dato>Mexico Mexico City Ciudad de Mexico de Mexico en el año 90 tenía 9815795 habitantes</dato>
<dato>Sao Paulo de Brazil en el año 96 tenía 9811776 habitantes</dato>
```



# Consultas XQuery

Ahora vamos a corregir los errores, ya que salen varios nombres para una ciudad.



The screenshot shows an XQuery editor interface. The left pane displays a file explorer for the path `C:/Users/joaqu/Downloads`, showing a directory structure with folders like `bin`, `data`, `etc`, `lib`, `repo`, `src`, and `webapp`, along with files like `BaseX.jar`, `CHANGELOG`, `LICENSE`, and `readme.txt`.

The main editor pane shows an XQuery script with the following lines:

```
1 for $ciudad in //city
2 let $nombre := $ciudad/name[last()]/string()
3 let $poblacion := $ciudad/population[last()]
4 let $pais := //country[@id=$ciudad/@country]/name[last()]/string()
5 order by $poblacion/number() descending
6 where $poblacion/number() > 9000000
7 return
8 <dato>{$nombre} de {$pais} en el año {$poblacion/@year/data()} tenía {$poblacion/
data()} habitantes</dato>
```

The status bar at the bottom of the editor indicates "Context: db:open('Ejemplo')", "2 : 36", and "OK".

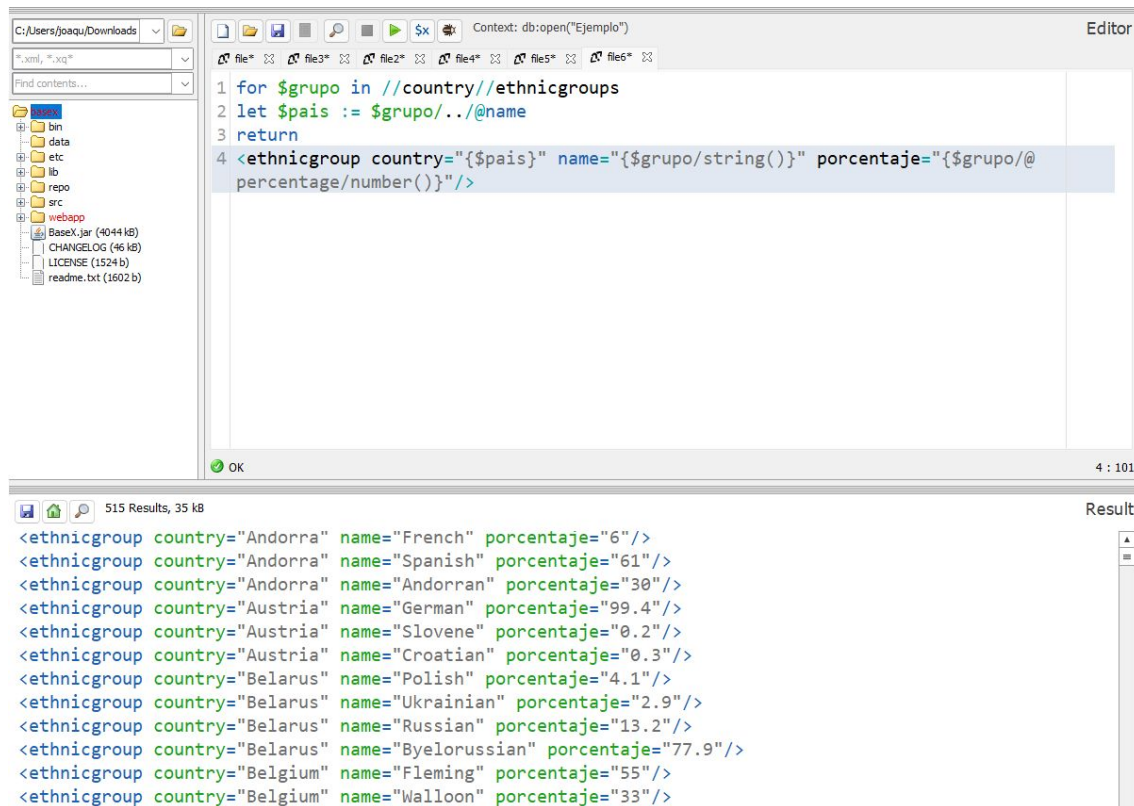
Below the editor, the results window shows "5 Results, 374 b" and displays the following XML output:

```
<dato>Seoul de South Korea en el año 95 tenía 10229262 habitantes</dato>
<dato>Greater de India en el año 91 tenía 9925891 habitantes</dato>
<dato>Karachi de Pakistan en el año 95 tenía 9863000 habitantes</dato>
<dato>Ciudad de Mexico de Mexico en el año 90 tenía 9815795 habitantes</dato>
<dato>Sao Paulo de Brazil en el año 96 tenía 9811776 habitantes</dato>
```

# Consultas XQuery

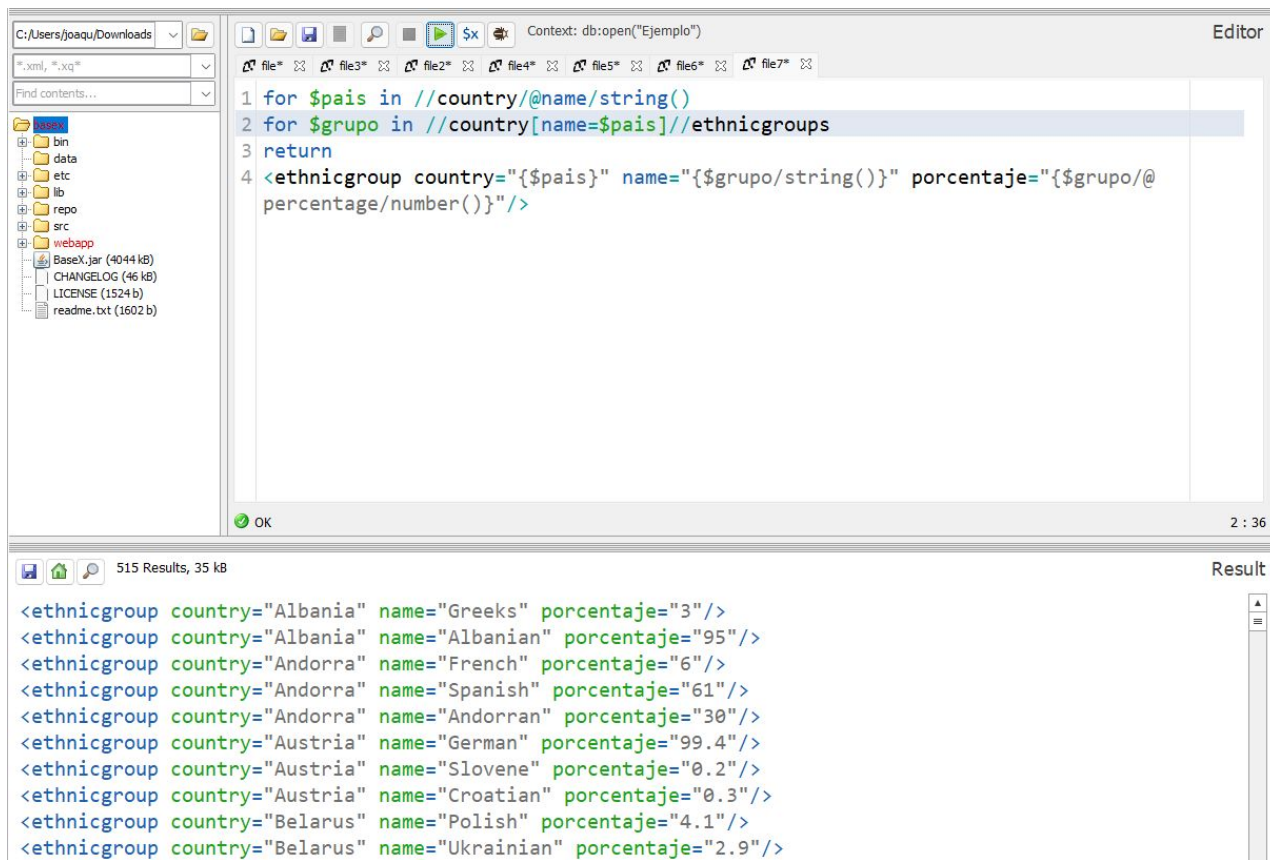
Vamos a obtener  
ahora los grupos  
étnicos existentes  
por país.

Se usa `//e1/..` para  
acceder a la  
etiqueta padre de  
`e1`



# For anidados en XQuery

Vamos a obtener ahora los grupos étnicos existentes por país usando for anidados, para que así nos aparezcan los datos agrupados por país en este caso.



The screenshot shows an XQuery editor window with the following code:

```
1 for $pais in //country/@name/string()  
2 for $grupo in //country[name=$pais]//ethnicgroups  
3 return  
4 <ethnicgroup country="{ $pais}" name="{ $grupo/string()}" porcentaje="{ $grupo/@  
percentage/number()}" />
```

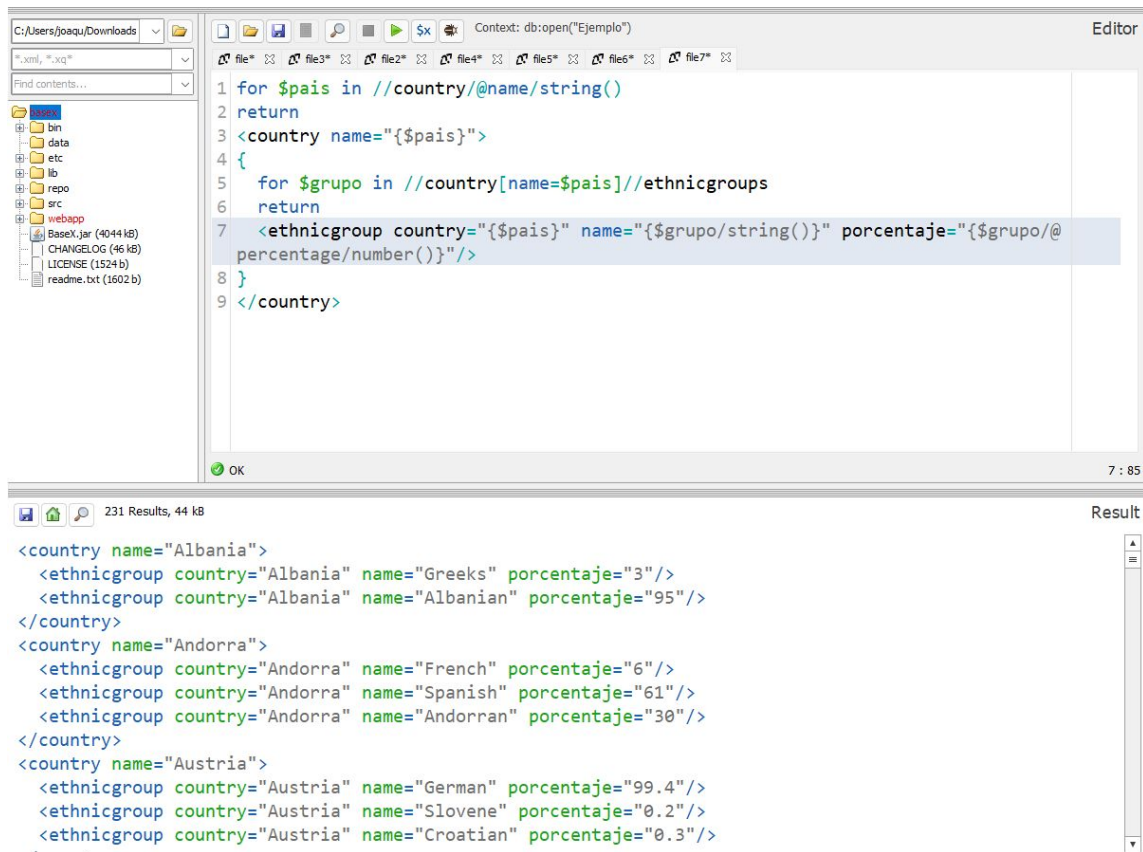
The editor window also shows a file explorer on the left with a directory structure including bin, data, etc, lib, repo, src, webapp, BaseX.jar, CHANGELOG, LICENSE, and readme.txt. The status bar at the bottom of the editor indicates "2 : 36".

Below the editor, the results viewer shows 515 Results, 35 kB. The results are displayed as XML fragments:

```
<ethnicgroup country="Albania" name="Greeks" porcentaje="3"/>  
<ethnicgroup country="Albania" name="Albanian" porcentaje="95"/>  
<ethnicgroup country="Andorra" name="French" porcentaje="6"/>  
<ethnicgroup country="Andorra" name="Spanish" porcentaje="61"/>  
<ethnicgroup country="Andorra" name="Andorran" porcentaje="30"/>  
<ethnicgroup country="Austria" name="German" porcentaje="99.4"/>  
<ethnicgroup country="Austria" name="Slovene" porcentaje="0.2"/>  
<ethnicgroup country="Austria" name="Croatian" porcentaje="0.3"/>  
<ethnicgroup country="Belarus" name="Polish" porcentaje="4.1"/>  
<ethnicgroup country="Belarus" name="Ukrainian" porcentaje="2.9"/>
```

# For anidados en XQuery

Vamos a obtener ahora los grupos étnicos existentes por país usando for anidados, para que nos aparezcan todas las etnias para cada país.



```
Context: db:open("Ejemplo")

1 for $pais in //country/@name/string()
2 return
3 <country name="{ $pais }">
4 {
5   for $grupo in //country[name=$pais]//ethnicgroups
6   return
7   <ethnicgroup country="{ $pais }" name="{ $grupo/string()}" porcentaje="{ $grupo/@
  porcentaje/number() }"/>
8 }
9 </country>
```

231 Results, 44 kB

```
<country name="Albania">
  <ethnicgroup country="Albania" name="Greeks" porcentaje="3"/>
  <ethnicgroup country="Albania" name="Albanian" porcentaje="95"/>
</country>
<country name="Andorra">
  <ethnicgroup country="Andorra" name="French" porcentaje="6"/>
  <ethnicgroup country="Andorra" name="Spanish" porcentaje="61"/>
  <ethnicgroup country="Andorra" name="Andorran" porcentaje="30"/>
</country>
<country name="Austria">
  <ethnicgroup country="Austria" name="German" porcentaje="99.4"/>
  <ethnicgroup country="Austria" name="Slovene" porcentaje="0.2"/>
  <ethnicgroup country="Austria" name="Croatian" porcentaje="0.3"/>
</country>
```