IMY 210
Practical 5: XQuery

XQuery (XML Query) is a query and functional programming language that queries and transforms collections of structured and unstructured data, usually in the form of XML, text and with vendor-specific extensions for other data formats (JSON, binary, etc.).

For this practical we will not only look at the GUI usage of the program BaseX to query semi-structure data, but also setup an XML server and learn to perform some simple server to server communicate with via PHP.

Files provided

- cd.xml – provided xml file that you will extract the data from

Tasks

Our XQuery

1. For this practical you will create several XQuery to transverse and retrieve data from the provided XML document. The queries can be directly inserted into a XQuery interface or saved as a .xq file to be executed later, for this practical we will do the later. The result of the query should only display the relevant data and not the associated nodes from the XML file.
    - **Basic selection and Arithmetic functions** – q1.xq
        i. Work out the total amount of money (**PRICE**) a customer will need to pay if they were to buy a copy of every CD displayed in the XML file, round off the final value if there are any decimal places

> **Note:** In many cases when executing a XQuery statement, the result will be rendered and interpreted as XML. This means, in some cases, the result presented should still follow the XML grammar and specification rulesets.

    - **Element construction** – q2.xq
        i. Display all the information of all CD published before (including) the **YEAR** 1987.

        ii. The format for the CD should be **ordered by** the **TITLE** and displayed in a html list.

         {Title} by {Artist} – {Company}, {Country}

        E.g.

         Red by The Communards – London, UK

        *Remember to include the <list> element as the root node

    - **Nested returns** – q3.xq
        i. Your final query should
            1. Count all the **CD** in the document
            2. Separate the **CD** by that cost (**PRICE**) more than the value of 9 and cost less than the value of 9.
            3. Count how many CD are in each category
            4. Sort this result by their **COUNTRY**.
            5. If there are several **CD** originating from the same **COUNTRY**, sort those **CD** within the **COUNTRY** by **TITLE**.
            6. Display only the **TITLE** of the **CD**

ii. The result will look like…

```
<TOTAL>26</TOTAL>
<ABOVE>
        <COUNT>11</COUNT>
        <TITLE> </TITLE> (more titles here)
</ABOVE>
<BELOW>
        <COUNT>15</COUNT>
        <TITLE> </TITLE> (more titles here)
</BELOW>
```

*Remember to include the <HTML> element as the root node

2. Remember to add your details to every XQuery file.
   XQuery comment format encapsulates the comment in (: :)
   e.g. **(:** This is a comment **:)**

Our XML server

3. You can simply run the BaseX program's GUI application and select "Database" to create a working server from existing XML files. The alternative (the way we will be using BaseX) is to run the **bat** script **basexhttp.bat** found inside the **bin** folder within the installation directory. After running the script, in a terminal, navigate to your **localhost** on the **port** indicated (check your terminal).

   From this point you can select the **Database Administrator** link and use the default **admin/admin** credentials to login. You can now connect to your very own XML database.

   Final notes:
   You can save any XML file (into the database) by placing them into the **bin** folder.
   You can save any query files for execution by placing them into the **webapp** folder.
   You can run any stored XQuery files by navigating to:
   http://localhost:port/rest/?run=queryfilename.xq

Our PHP script

4. You will often be task to create web applications that will talk to different services, this is even more so prominent in the current age of technology. With the **separation of concern** often not all services are hosted on the same server, this could be for multiple reasons such as security, efficiency and as failsafe measures. For this practical we will treat the XML server and PHP server as service hosted in different location.

   As oppose to using something like curl to carry out the operation we will be directly embedding the authentication information directly into the header of our http request. To complete this we will work with **streams**.

   > **Note:** Curl is a command line tool you will often see/use to transfer data to or from a server

   Generally working with steams, we will need to understand the following: context, data, parameter and the data being streamed. For this example, we will simply add the authentication information into our header request (the **context**).

Create a **p5.php** file and add the following.

```
array(
'http'=>array(
      'method' => "GET",
      'header' => "Authorization: Basic ".
      base64_encode("$username:$password")
      )
);
```

*Remember to set the **username** and **password** as a variable in the file

After creation this array we can set this to a variable and pass it to our steam's context

```
$context = stream_context_create($header);
```

Lastly, get the results;

```
$result = file_get_contents("\\URL", FALSE, $context);
```

*the \\URL will be our request the other server. Remember that URL access the query?

For this section you only need to request **q3**.xq. You will also only need to display the file you received in the results.

Bonus: Extra marks will be rewarded to present the result in XML format.

5. Submit your work on clickUP:
    - Compress all your **q#.xq** and **p5.php** into an archive named **P5.zip**.
    - Make a final backup of all your files.
    - Submit your ZIP file to the **Practical 5** assignment on the clickUP website before the end of **Monday**, **24 May**.

Good luck!