**5191 Project 5 Reflection Document**

Autumn Hood

This reflection should provide some insight into how I planned and executed the final deliverables for Project 2, for which I created a database of my actual record collection. I got the idea for my database when you mentioned your album collection database during one of our earlier classes. At first I was going to do my movies, but I decided that albums would have a more appropriate number and scope of elements; choosing my albums allowed me to build a database where each entity has 11 children, 9 grandchildren, and 1 great-grandchild.

**XML IA & tagging system**

I made any references to my collection consistent by using the word “album” throughout the document. I decided to not use the word “record” because I thought that hypothetically, that could create a problem in an XML database where common elements might include metadata tags such as “Record ID” or “Record #.” With <album\_collection> being my root tag, each separate entity is contained in the <album> tag, and all other elements are children of <album>.

Users typically have multiple methods of selecting an album, so I needed to provide these methods as elements of my database. The first and most important selection metric would be <artist> because so many people associate music with its performers, many people buy albums based on their artists, and “artist” is a quantitative, measurable way of describing the entities. Because I thought artist would be the most frequent search factor, it is the first column that appears on in the html output.

Additionally, many people develop specific searches around an <album\_title> and <track\_title>, so I included these as well. However, I couldn’t list the track titles at the same hierarchy as album titles and artist because the title is but one element of a larger picture: the track itself. Making the <track> parent tag allowed me to create other children besides <track\_title>, such as <track length>, <track\_number>, and <track\_credit>. Furthermore, people might know what an album looks like but not recall its name or artist, so I included the <album\_cover> to aid in visual searches.

People also frequently select music by its <genre> and by its <date\_released>. Yet many people only know the decade in which an album was released (sometimes the year), so I included information about the decade of release for searchability and information about the day, month, and year it was released for record-keeping purposes.

I included other metadata about the albums, record, such as <album\_length>, <album\_condition>, <record\_label>, and <number\_of\_tracks>.

**Code Explanation**

**XSLT & CSS from the 5191 course files**

As you know, I reverse-engineered your XSLT code from the “tecm5191” folder on github. When I first started, I just plugged my document names into your code to see if it would work. Of course, it didn’t. So I started over, and slowly began adding your lines of code in my document. Some of the most important information in the XSLT document lies before the <head> even begins, such as including the XMLNS (namespace) to avoid any conflicting elements, the <xsl:template match’””> so that the XSLT can find the document it uses to populate information into its <xsl:stylesheet>.

The other crucial elements of this document is, of course, the table, in which the <th>s match up chronologically, item-for-item with the <td>s within the body of the table. The <td>s are literally “selecting” (value of select=””) the element that you ask it to pull from the XML.

I ran into problems with the parent/child relationships of the “date\_released” and “album\_genre” elements, but the solution is relatively simple. You must tell the XSLT that *for each* <parent> element, you have to *select* the <child element>s, or simply <.> to populate all entries.

**Sortable table code**

After weeks that felt like years, I finally got my table to sort earlier today. I realized after somewhat of an epiphany that I was trying to apply the table to the wrong document, the .html. after talking with a classmate about an adjacent aspect of the project, I realized I needed to view the .xml document. I honestly don’t know why it took me so long (or even after going to OpenHack and getting expert insight) to get this. However, if there’s one thing I’ve learned from this project, it’s that the simplest mistakes are often the most terminal when working with markup langauges/code.

Funnily, the images are also sortable, and they merely sort according to the same values as they “artist” element. I have read how to eliminate their “sortability” online but haven’t put that method into practice.

I also reverse engineered your CSS to find out how to make my table have alternatingly colored rows. The only reason to apply this attribute would be

**Reflection**

I enjoyed this project. I definitely tried to challenge myself to learn more about the markup/ programming languages and functions required for this project. This project has been my first experience with some form of single-sourcing. In class, we’ve talked a lot about not teaching “tools,” a method with which I totally agree; but, I feel that this project actually gave us a chance to put into practice both the technology and tools required for rhetorically applying modular information (in a good way). I also enjoyed thinking about the UI and trying to implement a database that is both user-friendly and comprehensive.

Throughout this project, I stumbled upon websites, threads, forums, and solutions that indicate the sheer number of possibilities using these technologies. I almost don’t know how I’ve gotten nearly ¾ of my grad program without learning more about these technologies, but I suppose these topics don’t fit within any of the other courses offered. In any case, I’m glad I at least learned about it before I graduated. Please contact me with any questions or if I can explain anything more thoroughly.