DESARROLLO DE HOJA DE ESTILOS XSLT PARA FORMATEO Y EXTRACCIÓN DE INFORMACIÓN EN XML

Complemento de Bases de Datos 2020-2021 – 23/05/2021

Abstract

Los sistemas software muchas veces necesitan métodos para realizar la portabilidad de datos, por ejemplo, para garantizar el cumplimiento del Reglamento General de Protección de Datos de la Unión Europea. Para este propósito pueden emplearse las transformaciones mediante hojas de estilo XSL que permiten transformar los ficheros XML que ofrecen exportar algunos servicios. Este trabajo investiga la sintaxis necesaria para el desarrollo de hojas de estilo sobre un ejemplo de uso real y desarrolla una serie de hojas de estilo que permiten aplicarse sobre el ejemplo expuesto.

Aguilar Alhama, Andrés
andagualh@alum.us.es
Moreno Delgado, Pablo
pabmordel@alum.us.es

Indice de Contenidos

1. Introducción

Acorde al título del trabajo: "Desarrollo de una hoja de estilos XSLT para formateo y extracción de información en XML", con objetivo de investigar la tecnología XSLT y su función en la exportación y formateo de datos extraídos de una base de datos, una página web o una API (En español: Interfaz de Programación de Aplicaciones). XML y XSLT son lenguajes mantenidos por el *World Wide Web Consortium* (Con siglas W3C) como estándares para el procesado de datos de la Web.

Para este propósito, hemos escogido el resultado de la exportación de los datos del sitio web *MyAnimeList* y hemos decidido convertir el contenido tanto en una página HTML como en un script SQL con el objetivo de portar los datos contenidos en un sistema de bases de datos propio.

En el contexto de la asignatura Complemento de Bases de Datos (CBD), la temática de este trabajo está relacionada con el temario de *Web Semántica*, en el se ha hecho una mención breve a las transformaciones XSLT y el lenguaje de marcado XML.

1.1. XML

XML se define como un metalenguaje para lenguajes de marcas, su nombre completo es *eXtensible Markup Language* (En español: Lenguaje de Marcado Extensible). Su desarrollo se realizó en respuesta a las páginas HTML de la web temprana y su inconsistente estructura. Dado que las páginas HTML no suelen estar estructuradas siempre de la misma forma, siguiendo formatos que varían enormemente, muchas veces se debe lidiar con muchas ambigüedades a la hora de tratar sus datos de forma automática. (Varios, Wikipedia, 2021)

El formato XML es el hijo de otro lenguaje conocido como SGML (En español: Lenguaje Estándar de Marcado Generalizado), añadiendo nuevas características: Extensibilidad (eXtensible), nuevos analizadores más simples y no aceptar documentos que contengan errores en su sintaxis. (Varios, Wikipedia, 2021)

Por tanto, este será el formato que se utilizará para el fichero de entrada del que obtendremos los datos que procesaremos para el Ejemplo de Uso.

1.2. XSLT

Para hablar de XSLT primero debemos mencionar la familia de lenguajes XSL. XSL hace referencia a las siglas de *eXtensible Stylesheet Language* (En español: Lenguaje Extensible de Hojas de Estilo). Esta familia de lenguajes define como se presentan los datos extraídos de un documento XML. Es dentro de este conjunto que encontramos lo que se conoce como transformaciones XSL (Definido por sus siglas XSLT). (Varios, Wikipedia, 2021)

Las transformaciones XSL consisten en una hoja de estilo cuya función es transformar ficheros XML en otros formatos como HTML o XHTML con el propósito de extraer, filtrar y presentar los datos contenidos de un fichero de origen. (Varios, Wikipedia, 2021)

Para familiarizarnos con la sintaxis de las transformaciones XSL, hemos hecho uso de las siguientes referencias:

- (W3School, s.f.)
- (Stein, s.f.)

1.3. Ejemplo de Uso: Contexto

Con el fin de poner un ejemplo práctico de uso, hemos extraído una lista en formato XML del sitio web "*MyAnimeList*", que permite exportar los datos de una lista de visionado de series de animación japonesa.

Esta lista de visionado contiene un registro creado por el usuario de las series y películas que ha visto, estos datos corresponden a los títulos, capítulos totales de la serie (1, en el caso de las películas), capítulos visionados, estado del visionado por parte del usuario (Actualmente en visionado, completado, dejado o planificado para visionado) y comentarios establecidos por el usuario; entre otros.

Los datos exportados en XML permiten su fácil inclusión en otros sitios web, como pueden ser *AniList* o *Kitsu dot io*. Para el propósito de formateado y filtración, la tecnología XSLT puede ayudarnos a importar estos datos en un servicio nuevo.

Un ejemplo real fue durante 2018, cuando el sitio web en el que se aloja dicha lista pasó por un gran mantenimiento que duró meses. (Orsini, 2018) (Peters, 2018) Durante este periodo, muchos usuarios se vieron en la necesidad de exportar sus listas en XML para poder importarlos en otros servicios similares.

1.4 Objetivos del proyecto

Este proyecto tiene como objetivo principal la familiarización con el lenguaje xsl y las transformaciones que este puede realizar sobre los ficheros xml.

Objetivos más específicos del proyecto, conciernen a la fácil exportación de los datos contenidos en el fichero y su transformación en otros formatos que sean fácilmente reconocido por otros sistemas, ya sean bases de datos basadas en documentos o bases de datos basadas en el lenguaje SQL. Entre nuestros objetivos, hemos optado por formatear los datos para su inclusión en una base de datos SQL, aunque también hemos realizado pruebas para exponer los datos en HTML.

1.5. Entorno de Desarrollo y Herramientas empleadas

El desarrollo del trabajo ha empleado diversas herramientas, comenzando por un procesador de transformaciones XSL como es **Saxon**.

Saxon se encuentra disponible en muchas versiones, sin embargo, nuestra elección ha sido su implementación en un sitio web disponible en la siguiente referencia (XSLTTest, s.f.). La versión implementada de Saxon cuenta con soporte para XSLT 2.0 y permite realizar transformaciones además de mostrarlas interpretadas si se están realizando en HTML. El citado sitio web ha sido usado para realizar las transformaciones y probar los resultados que se han realizado en HTML.

Otra de las herramientas que se ha empleado es **Oracle Database 11g Express Edition** junto a **SQL Developer**. Dado que nuestro interés a la hora de realizar transformaciones sobre el fichero XML escogido ha sido para importarlo en una base de datos SQL, hemos empleado Oracle Database 11g como base de datos por su facilidad de uso y SQL Developer como entorno de desarrollo para probar y debuggear el resultado de las transformaciones a formato SQL.

El principal motivo para el uso de estas tecnologías no es más que su simplicidad y facilidad de uso, especialmente de Oracle Database 11g y SQLDeveloper al haber sido ampliamente usadas en la presente asignatura durante el transcurso de las prácticas.

También se ha utilizado Github para alojar el repositorio que contiene el fichero xml de prueba y las hojas de estilo desarrolladas. El repositorio Github realizado para este proyecto se puede encontrar en la siguiente referencia (Andagualh, s.f.).

2. Desarrollo del Trabajo

2.1. El fichero de entrada XML

Primero debemos conocer cómo se estructura un fichero XML. La estructura de un fichero XML viene dada por nodos, que pueden contener más nodos en su interior. Técnicamente todo lo que se contiene en un fichero XML se denomina nodo, existen varios tipos de nodos:

- El documento en sí, se considera un nodo de tipo documento. Se encuentran al principio del documento empleando la siguiente sintaxis: *<?xml versión="[VERSION]" encoding="[ENCODING-FORMAT]"?>*
- Los elementos de la sintaxis XML de denominan nodo de elemento. Se notan en la sintaxis entre los siguientes símbolos *<NombreDelNodo>*.
- El texto que podemos encontrar en el documento se denominan nodos de texto.
- Los atributos contenidos en el fichero se consideran nodos de atributo.
- Los comentarios marcados con la sintaxis *<!—ContenidoComentario -->* se denominan nodos de comentario.

Los nodos en un documento XML pueden contener otros nodos en su interior, formando así la estructura de los datos.

A continuación, podemos ver una porción de nuestro fichero XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--
 Created by XML Export feature at MyAnimeList.net
 Version 1.1.0
-->

<myanimelist>

    <myinfo>
        <user_id>3256291</user_id>
        <user_name>Acofmelt</user_name>
        <user_export_type>1</user_export_type>
        <user_total_anime>332</user_total_anime>
        <user_total_watching>6</user_total_watching>
        <user_total_completed>307</user_total_completed>
        <user_total_onhold>9</user_total_onhold>
        <user_total_dropped>9</user_total_dropped>
        <user_total_plantowatch>1</user_total_plantowatch>
    </myinfo>


        <anime>
            <series_animedb_id>35248</series_animedb_id>
            <series_title><![CDATA[18if]]></series_title>
            <series_type>TV</series_type>
            <series_episodes>13</series_episodes>
            <my_id>0</my_id>
            <my_watched_episodes>5</my_watched_episodes>
            <my_start_date>0000-00-00</my_start_date>
            <my_finish_date>0000-00-00</my_finish_date>
            <my_rated></my_rated>
            <my_score>0</my_score>
            <my_storage></my_storage>
            <my_storage_value>0.00</my_storage_value>
            <my_status>Dropped</my_status>
            <my_comments><![CDATA[No tenia tiempo pa verla, perdí el interés.]]></my_comments>
            <my_times_watched>0</my_times_watched>
            <my_rewatch_value></my_rewatch_value>
            <my_priority>LOW</my_priority>
            <my_tags><![CDATA[]]></my_tags>
            <my_rewatching>0</my_rewatching>
            <my_rewatching_ep>0</my_rewatching_ep>
            <my_discuss>1</my_discuss>
            <my_sns>default</my_sns>
            <update_on_import>0</update_on_import>
        </anime>
```

En la captura expuesta, podemos identificar distintos tipos de nodos:

- El nodo "*myanimelist*" es el nodo raíz que contiene el resto de los nodos que contienen datos en el documento, este nodo sólo cuenta con dos hijos: "*myinfo*" y "*anime*". Es equivalente en SQL al nombre de la base de datos.
- El nodo "*myinfo*" contiene diversa información y estadísticas de la cuenta de la que se ha extraído la lista de series y películas. Estos nodos contienen información como el id interno del usuario en el sitio web original, el nombre de usuario o el número de elementos total de la lista. Es equivalente en SQL al nombre de la tabla.
    - Los nodos contenidos en "*myinfo*" son de tipo elemento.
    - Los nodos de tipo elemento contenidos en "*myinfo*" tienen en su interior nodos de texto. Los nodos tipo elemento son equivalentes en SQL a las columnas de una tabla.
- El nodo "*anime*" contiene la información relativa a una entrada en la lista de series y películas. Esta información puede ser el número de identificación de identificación en la base de datos origen, el título de la entrada, el número de episodios total, el número de episodios vistos o las fechas en las que se ha visionado el medio audiovisual asociado a esa entrada.
    - Los nodos contenidos en "*anime*" son de tipo texto.
    - Los nodos de tipo elemento contenidos en "*anime*" tienen en su interior nodos de texto. Los nodos tipo elemento son equivalentes en SQL a las columnas de una tabla.
- Hay que destacar que la exportación no cuenta con ningún nodo de tipo atributo, sólo el nodo raíz, nodos de elemento y nodos de texto. Estamos excluyendo de la cuenta anterior el nodo de comentario y el nodo de documento que se encuentra en el fichero, ya que no afectan a las transformaciones que se van a realizar.

2.2. Sintaxis Básica – Hoja de estilos Simple

Para llevar a cabo una primera toma de contacto con el lenguaje, se decidió como objetivo crear una plantilla que convirtiera el XML en cuestión en una lista HTML que mostrara la colección completa de series y películas, así como sus atributos más importantes, además de los datos de la cuenta personal del usuario.

Para ello, no hizo falta implementar funciones complejas, pues sólo había que obtener los parámetros buscados a partir de su identificador. Esto puede llevarse a cabo mediante "*<xsl:value-of select="identificador"/>*", que combinado con un poco de sintaxis HTML nos dará como resultado el siguiente código:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="myinfo">
<div style='text-align:center'><u><b> Datos de usuario </b></u></div>
<br></br>
<fieldset>
<p>Nombre de usuario: <xsl:value-of select="user_name"/></p>
<p>Animes totales: <xsl:value-of select="user_total_anime"/></p>
<p>Animes viendo: <xsl:value-of select="user_total_watching"/></p>
<p>Animes vistos: <xsl:value-of select="user_total_completed"/></p>
<p>Animes en espera: <xsl:value-of select="user_total_onhold"/></p>
<p>Animes dejados: <xsl:value-of select="user_total_dropped"/></p>
<p>Animes pendientes: <xsl:value-of select="user_total_plantowatch"/></p>
</fieldset>
<br></br>
<br></br>
<div style='text-align:center'><u><b> Lista de animes </b></u></div>
<br></br>
  </xsl:template>

 <xsl:template match="anime">
<fieldset>
<b><p><div style='text-align:center'><xsl:value-of select="series_title"/></div></p></b>
<p>Formato: <xsl:value-of select="series_type"/></p>
<p>Episodios totales: <xsl:value-of select="series_episodes"/></p>
<p>Episodios vistos: <xsl:value-of select="my_watched_episodes"/></p>
<p>Mi puntuación: <xsl:value-of select="my_score"/></p>
<p>Estado: <xsl:value-of select="my_status"/></p>
<p>Comentarios: <xsl:value-of select="my_comments"/></p>
</fieldset>
  </xsl:template>

</xsl:stylesheet>
```

La hoja de estilos empleada puede encontrarse en la hoja de estilos presente en la ruta "*stylesheet/Lista_general.xsl*" del repositorio, así como en la entrega bajo el mismo nombre de archivo.

Esta plantilla nos proporcionará la lista buscada, gracias a los parámetros HTML "*<p>*", "*<fieldset>*" y "*
*" nos devolverá el resultado en un formato legible y atractivo para consultar la información que nos interese. De este modo, al ejecutarlo sobre el XML en cuestión obtendremos la siguiente lista:

Datos de usuario

Nombre de usuario: Acofmelt
Animes totales: 332
Animes viendo: 6
Animes vistos: 307
Animes en espera: 9
Animes dejados: 9
Animes pendientes: 1

Lista de animes

18if
Formato: TV
Episodios totales: 13
Episodios vistos: 5
Mi puntuación: 0
Estado: Dropped
Comentarios: No tenia tiempo pa verla, perdí el interés.

3-gatsu no Lion
Formato: TV
Episodios totales: 22
Episodios vistos: 22
Mi puntuación: 7
Estado: Completed
Comentarios:

Así pues, finalmente obtenemos de forma sencilla el resultado esperado mediante la básica obtención de parámetros y formateo HMTL, para a continuación adentrarnos un poco más en las funcionalidades existentes en XSLT.

2.3. Sintaxis de Filtrado – Hoja de estilo que filtra por estado

Con objetivo de adentrarnos un poco más en el lenguaje en cuestión, se llevó a cabo un filtrado de las series para solo mostrar aquellas pertenecientes a la lista que no estén acabadas por el usuario, además de indicarle cuántos episodios le quedan para finalizarla.

Para llevar a cabo el filtrado, se ha usado la funcionalidad "*if*" del lenguaje, en pos de sólo mostrar aquellos que no estén finalizados. De este modo, la plantilla obtenida es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="myinfo">
<div style='text-align:center'><u><b> Datos de usuario </b></u></div>
<br></br>
<fieldset>
<p>Nombre de usuario: <xsl:value-of select="user_name"/></p>
<p>Animes totales: <xsl:value-of select="user_total_anime"/></p>
<p>Animes viendo: <xsl:value-of select="user_total_watching"/></p>
<p>Animes vistos: <xsl:value-of select="user_total_completed"/></p>
<p>Animes en espera: <xsl:value-of select="user_total_onhold"/></p>
<p>Animes dejados: <xsl:value-of select="user_total_dropped"/></p>
<p>Animes pendientes: <xsl:value-of select="user_total_plantowatch"/></p>
</fieldset>
<br></br>
<br></br>
<div style='text-align:center'><u><b> Animes sin terminar </b></u></div>
<br></br>
  </xsl:template>

 <xsl:template match="anime">
<xsl:if test="my_status!='Completed'">
<fieldset>
Aún no has terminado  <b><xsl:value-of select="series_title"/></b>, has visto <b><xsl:value-of select="my_watched_episodes"/>
</b> de sus <b><xsl:value-of select="series_episodes"/></b> episodios.
</fieldset>
</xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

La hoja de estilos empleada puede encontrarse en la hoja de estilos presente en la ruta "*stylesheet/Lista_animes_a_medias.xsl*" del repositorio, así como en la entrega bajo el mismo nombre de archivo.

Así pues, usando algo de HTML para que el resultado se muestre en un formato apropiado, obtendremos el siguiente resultado indicando qué series existen sin finalizar y cuántos capítulos quedan por ver para ello.

Datos de usuario

Nombre de usuario: Acofmelt

Animes totales: 332

Animes viendo: 6

Animes vistos: 307

Animes en espera: 9

Animes dejados: 9

Animes pendientes: 1

Animes sin terminar

Aún no has terminado **18if**, has visto **5** de sus **13** episodios.

Aún no has terminado **3-gatsu no Lion 2nd Season**, has visto **4** de sus **22** episodios.

Aún no has terminado **91 Days**, has visto **1** de sus **12** episodios.

Aún no has terminado **Beatless**, has visto **8** de sus **20** episodios.

Aún no has terminado **Boku no Hero Academia 3rd Season**, has visto **15** de sus **25** episodios.

Aún no has terminado **Citrus**, has visto **9** de sus **12** episodios.

Aún no has terminado **Gamers!**, has visto **11** de sus **12** episodios.

Aún no has terminado **Gangsta.**, has visto **4** de sus **12** episodios.

Aún no has terminado **Gundam: G no Reconguista**, has visto **7** de sus **26** episodios.

Aún no has terminado **Hakata Tonkotsu Ramens**, has visto **7** de sus **12** episodios.

Finalmente observamos que el filtrado mediante "*if*" se ha efectuado correctamente y los datos que se muestran son los que buscábamos, por lo que a continuación continuaremos profundizando un poco más en este lenguaje.

2.4. Sintaxis de Filtrado – Hoja de estilo que filtra por puntuación

Nuestro siguiente objetivo es profundizar un poco más en los recursos que ofrece este lenguaje mediante el uso del condicional "*when*" que nos permitirá aplicar diversas acciones cuando se cumplan los requisitos especificados. En este caso, usaremos dicho recurso para categorizar las series y películas aplicando diferentes colores según la nota que el usuario le haya puesto a cada uno. Para ello se ha usado una escala de rojo (si su nota es un cero) a verde (si su nota es un diez).

Para ello, se ha usado el condicional "*when*" para tratar cada una de las notas posibles y así aplicarle el color que le corresponda, tal como se puede ver a continuación.

```
<xsl:choose>
<xsl:when test="my_score=10">
<b><p style="color:#28FF00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=9">
<b><p style="color:#5EFF00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=8">
<b><p style="color:#9BFF00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=7">
<b><p style="color:#E1FF00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=6">
<b><p style="color:#FFFE00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=5">
<b><p style="color:#FFCC00"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=4">
<b><p style="color:#FFA200"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=3">
<b><p style="color:#FF6400"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=2">
<b><p style="color:#FF4200"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=1">
<b><p style="color:#FF1700"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:when test="my_score=0">
<b><p style="color:#9E0000"><xsl:value-of select="series_title"/></p></b>
</xsl:when>
<xsl:otherwise>
            <b><p><div style='text-align:center'><xsl:value-of select="series_title"/></div></p></b>
        </xsl:otherwise>
  </xsl:choose>
```

La hoja de estilos empleada puede encontrarse en la hoja de estilos presente en la ruta "*stylesheet/Lista_animes_por_colores.xsl*" del repositorio, así como en la entrega bajo el mismo nombre de archivo. Por temas de tamaño de imagen, sólo hemos añadido la parte más interesante del código que queremos recalcar en este caso, para ver la hoja de estilos completa, puede consultarla en el enlace indicado.

Tal como se puede observar, se ha usado un código RGB hexadecimal para aplicarle un color diferente según la nota que posea el elemento. Una vez ejecutada esta plantilla, obtendremos la lista general del usuario, pero los nombres de las series y películas aparecerán con su color correspondiente.

Una vez aprendido los aspectos básicos de este lenguaje y habiendo conseguido obtener los resultados esperados, avanzaremos un poco más para conseguir un objetivo más ambicioso: pasar los datos existentes en el XML a una base de datos SQL.

2.5. Portando el contenido a SQL – Hoja de estilo con script SQL como resultado

Una vez aprendido los aspectos básicos de XSLT, nuestro objetivo es crear una hoja de estilo con script SQL a partir del XML con el que estamos trabajando. De este modo, podríamos introducir de manera casi inmediata todos los datos que nos proporciona el XML en una base de datos SQL para así poder hacer uso de todos los recursos que una base de datos como esta proporciona.

Usando lo aprendido en las plantillas anteriores, obtendremos los parámetros de los elementos de la lista y mediante "*CREATE TABLE*" e "*INSERT INTO*" se podrá llevar a cabo una sencilla plantilla que produzca la hoja de estilo SQL que buscamos.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="myinfo">
CREATE TABLE user (
userId int,
username varchar(255),
totalanime int,
totalwatching int,
totalcompleted int,
totalonhold int,
totaldropped int,
totalplantowatch int);

INSERT INTO user (userId, username, totalanime, totalwatching, totalcompleted, totalonhold, totaldropped, totalplantowatch) VALUES (
<xsl:value-of select="user_id"/> , '<xsl:value-of select="user_name"/>' ,
<xsl:value-of select="user_total_anime"/> , <xsl:value-of select="user_total_watching"/> ,
<xsl:value-of select="user_total_completed"/> , <xsl:value-of select="user_total_onhold"/> ,
<xsl:value-of select="user_total_dropped"/> , <xsl:value-of select="user_total_plantowatch"/> );

CREATE TABLE anime(
animeId int,
title varchar(255),
type varchar(255),
episodes int,
watchedepisodes int,
score int,
comments varchar(255));
  </xsl:template>

<xsl:template match="anime">

INSERT INTO anime (animeId, title, type, episodes, watchedepisodes, score, comments) VALUES (
<xsl:value-of select="series_animedb_id"/> , '<xsl:value-of select="series_title"/>' ,
'<xsl:value-of select="series_type"/>' , <xsl:value-of select="series_episodes"/> ,
<xsl:value-of select="my_watched_episodes"/> , <xsl:value-of select="my_score"/> ,
'<xsl:value-of select="my_comments"/>');

  </xsl:template>

</xsl:stylesheet>
```

La hoja de estilos empleada puede encontrarse en la hoja de estilos presente en la ruta "*stylesheet/Lista_animes_SQL.xsl*" del repositorio, así como en la entrega bajo el mismo nombre de archivo.

Tal como podemos observar, no se requiere ningún conocimiento complejo de este lenguaje para llevar a cabo esta plantilla, pero nos otorgará el resultado esperado para poder introducir todos los datos que nos interesan en una base de datos SQL si así lo deseáramos.

```
<?xml version="1.0" encoding="UTF-8"?>


CREATE TABLE user (
userId int,
username varchar(255),
totalanime int,
totalwatching int,
totalcompleted int,
totalonhold int,
totaldropped int,
totalplantowatch int);

INSERT INTO user (userId, username, totalanime, totalwatching, totalcompleted, totalonhold, totaldropped, totalplantowatch) VALUES (
3256291 , 'Acofmelt' ,
332 , 6 ,
307 , 9 ,
9 , 1 );

CREATE TABLE anime(
animeId int,
title varchar(255),
type varchar(255),
episodes int,
watchedepisodes int,
score int,
comments varchar(255));




INSERT INTO anime (animeId, title, type, episodes, watchedepisodes, score, comments) VALUES (
35248 , '18if' ,
'TV' , 13 ,
5 , 0 ,
'No tenia tiempo pa verla, perdí el interés.');




INSERT INTO anime (animeId, title, type, episodes, watchedepisodes, score, comments) VALUES (
31646 , '3-gatsu no Lion' ,
'TV' , 22 ,
22 , 7 ,
'');
```

Así pues, obtendremos la hoja de estilo buscada y podremos ejecutarla en nuestra base de datos SQL para introducir todos los datos deseados que nos proporciona el XML en cuestión. Por último, nos adentraremos un paso más en las posibilidades que nos otorga XSLT llevando a cabo una plantilla genérica que nos proporcione la hoja de estilo SQL que buscamos, pero sin introducir los nombres concretos de las variables, para así poder usarla de forma genérica siempre que se cumplan ciertos requisitos.

Para probar el resultado, lo hemos introducido en *SQL Developer* para comprobar su funcionamiento. Podemos observar el estado de la tabla tras la ejecución:

ANIMEID	TITLE	TYPE	EPISODES	WATCHEDEPISODES	SCORE	COMMENTS
35248	18if	TV	13	5	0	No tenia tiempo pa verla, perdí el interés.
31646	3-gatsu no Lion	TV	22	22	7	(null)
35180	3-gatsu no Lion 2nd Season	TV	22	4	0	(null)
32998	91 Days	TV	12	1	0	(null)
36039	A.I.C.O.: Incarnation	ONA	12	12	8	(null)
33337	ACCA: 13-ku Kansatsu-ka	TV	12	12	10	(null)
11759	Accel World	TV	24	24	7	(null)
13939	Accel World EX	OVA	2	2	6	(null)
31790	Active Raid: Kidou Kyoushuushitsu Dai Hachi Gakari	TV	12	12	7	(null)
32301	Active Raid: Kidou Kyoushuushitsu Dai Hachi Gakari 2nd	TV	12	12	8	(null)
31580	Ajin	TV	13	13	8	(null)
22729	Aldnoah.Zero	TV	12	12	8	(null)
27655	Aldnoah.Zero 2nd Season	TV	12	12	8	(null)
6547	Angel Beats!	TV	13	13	8	(null)
9989	Ano Hi Mita Hana no Namae wo Bokutachi wa Mada Shiranai.	TV	11	11	7	(null)
11111	Another	TV	12	12	6	Downloaded Episodes: 12
7647	Arakawa Under the Bridge	TV	13	13	7	(null)
33371	Atom: The Beginning	TV	12	12	8	(null)
32827	B: The Beginning	ONA	12	12	8	(null)
2251	Baccano!	TV	13	13	8	(null)
3901	Baccano! Specials	Special	3	3	8	(null)
5081	Bakemonogatari	TV	15	15	9	(null)
1589	Bartender	TV	11	11	10	(null)
20543	Bayonetta: Bloody Fate	Movie	1	1	6	Downloaded Episodes: 1
36516	Beatless	TV	20	8	0	(null)
57	Beck	TV	26	26	9	(null)
31904	Big Order (TV)	TV	10	10	7	(null)

Como se puede ver, los datos han sido introducidos en la tabla de manera correcta.

2.6. Funciones – Hoja de estilo recursiva que reconoce cualquier dato introducido

Uno de los principales problemas que hemos encontrado desarrollando las cuatro hojas de estilo anteriores, ha sido que teníamos que conocer los campos específicos con los que contaba el fichero de entrada XML. Por tanto, hemos añadido el uso de dos funciones conocidas como "*node()*" y "*name()*".

La función *node()*, al ser llamada devolverá el contenido del nodo que estamos indicando. El uso de esta función es una de las que nos permiten extraer las etiquetas de metalenguaje que marcan la estructura del fichero XML.

La función name(), hará referencia al nombre que tiene el propio nodo sobre el que nos encontramos.

Añadir, ya que los nodos se disponen en estructura tipo árbol podemos acceder a ellos notándolo en la sintaxis como una ruta. Por ejemplo, para nuestro XML si queremos acceder a los nodos "*myinfo*" y "*anime*", debemos indicarlo de la siguiente manera: "*myanimelist/node()*", situándonos en el contenido del nodo "*myanimelist*" que son los nodos "*myinfo*" y "*anime*".

Para comenzar a usar estas funciones, hemos realizado una hoja de estilo que toma el contenido de los nodos presentes en el nodo raíz "*myanimelist*" y los formatea en lenguaje de marcado HTML. La hoja de estilos se puede observar en la siguiente captura:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="myanimelist/node()">
<xsl:if test="node() != ' ' ">
<fieldset>
<p> Tabla: <xsl:value-of select="name()"/></p>
<xsl:for-each select="*">


<p><xsl:value-of select="name()"/>: <xsl:value-of select="node()"/></p>

</xsl:for-each>
</fieldset>
</xsl:if>

  </xsl:template>

</xsl:stylesheet>
```

Es una hoja de estilos muy similar a las realizadas anteriormente, pero podemos ver algunas diferencias.

La primera diferencia es la que indica el template, indicando al interprete que queremos trabajar sobre el contenido del nodo "*myanimelist*".

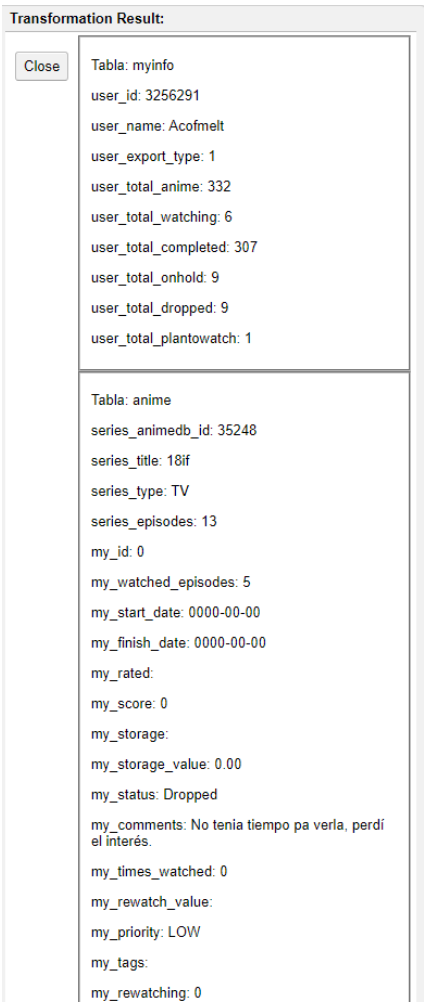
A continuación, definimos una condición if para excluir posibles nodos vacíos ya que como no tienen datos no nos interesan. La decisión de incluir este if ha sido en respuesta a uno de los problemas encontrados: La hoja de estilo reconocía nodos que estaban vacío, seguramente los saltos de línea que se encuentran en el fichero XML.

Una vez excluidos los nodos con contenido vacío, seleccionamos el atributo *name()* de los nodos contenidos en "*myanimelist*". En este caso se trata de las cadenas de texto "*myinfo*" y "anime".

A continuación emplazamos un bucle for-each para recorrer todo el contenido de los nodos "*myinfo*" y "*anime*", de los que extraemos el nombre del nodo de elemento con *name()* y su contenido con *node()*.

Una porción del resultado de la transformación realizada sobre el fichero XML es el siguiente:

A continuación, vemos una porción del resultado en un navegador que pueda interpretar el marcado HTML:



Como vemos la transformación ha sido capaz de extraer tanto el nombre de la tabla en la base de datos origen como las columnas, las cuales se encuentran asociadas a su contenido de texto.

La hoja de estilos empleada puede encontrarse en la hoja de estilos presente en la ruta "*stylesheet/HTML Ejemplo Generico.xsl*" del repositorio, así como en la entrega bajo el mismo nombre de archivo.

El resultado se puede encontrar en la ruta "*results/HTMLGenerico.html*" en el repositorio, así como en la entrega bajo el mismo nombre de archivo.

2.5. Sintaxis Avanzada y XSLT 2.0 – Hoja de estilo recursiva con script SQL como resultado

La hoja de estilos del apartado anterior nos ha permitido acceder a los nodos sin importar cuales fuesen sus nombres, automatizando el proceso de formato un paso más. Entonces, nuestro siguiente paso fue emplear las funciones "*node()*" y "*name()*" para generar un script SQL que permita exportar los datos.

Sin embargo, a la hora de elaborar esta clase de hojas de estilos nos topamos con un problema importante: La sintaxis SQL requiere denotar los datos por distintos tipos a la hora de introducirlos en una tabla. Con esto queremos decir que se distingue entre si un dato es un número entero, una cadena de texto o una fecha.

Tras investigar la sintaxis de XSLT 1.0, concluimos que no podíamos hacer dicha distinción en esta versión de XSLT, por lo que empezamos a utilizar XSLT 2.0 para esta hoja de estilos. XSLT 2.0 incluye nuevas funciones y sentencias que podemos usar, en nuestro caso aquella que las necesidades del proyecto requerían es la función "*castable* as". Esta función intenta *castear* (Interpretar) el dato que se le está pasando como el tipo que se haya definido en la sintaxis.

Nuestro fichero XML cuenta con datos de los tipos *integer* (número entero), *double* (números con decimales), *date* (fechas) y *string* (Cadenas de Texto). Por comodidad, hemos definido solo la condición para distinguir entre *integer* y *string*. Los nodos de texto tipo *double* y tipo *date* se interpretarán como *string*.

Otro de los problemas encontrados ha sido a la hora de crear el script para las tablas, lamentablemente este problema ha quedado sin resolver. El problema presentado ha quedado sin resolver debido a complicaciones con la sintaxis.

Por ejemplo, en las transformaciones XSL no se pueden definir dos templates que intenten encajar con la misma ruta de nodos. Esto se debe a que saltará un error, sin embargo, si el procesador usado no indica el error debe recuperarse escogiendo entre los templates que tiene disponible (Roberts, 2013). En nuestro caso, siempre se escogía el segundo template durante las ejecuciones.

Para circunventar este problema, hemos tenido que definir las tablas a mano con el siguiente script a la hora de construir la estructura de la base de datos:

```sql
DROP TABLE MYINFO;
DROP TABLE ANIME;

CREATE TABLE myinfo (
  user_id int,
  user_name varchar(255),
  user_export_type int,
  user_total_anime int,
  user_total_watching int,
  user_total_completed int,
  user_total_onhold int,
  user_total_dropped int,
  user_total_plantowatch int
);

CREATE TABLE anime(
    series_animedb_id int,
    series_title varchar(255),
    series_type varchar(255),
    series_episodes int,
    my_id int,
    my_watched_episodes int,
    my_start_date varchar(255),
    my_finish_date varchar(255),
    my_rated int,
    my_score int,
    my_storage varchar(255),
    my_storage_value varchar(255),
    my_status varchar(255),
    my_comments varchar(255),
    my_times_watched int,
    my_rewatch_value varchar(255),
    my_priority varchar(255),
    my_tags varchar(255),
    my_rewatching int,
    my_rewatching_ep int,
    my_discuss int,
    my_sns varchar(255),
    update_on_import int);
```

El siguiente script puede encontrarse en el repositorio en la ruta "*stylesheets/Tablas Completas.sql*" o bajo el mismo nombre de archivo en los documentos de la entrega.

A continuación, observamos la hoja de estilos desarrollada tras haber abordado, no siempre con éxito, los distintos problemas descritos.

```xml
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsl:output method="text" encoding="UTF-8" indent="yes" />


<xsl:template match="myanimelist/node()">
    <xsl:if test="node() != ' ' ">
INSERT INTO <xsl:value-of select="name()"/>(<xsl:for-each select="*">
                    <xsl:if test="node()/name() != ' '">
                    <xsl:value-of select="name()"/>
                    <xsl:if test='position() != last()'>,</xsl:if>
                    </xsl:if>
                </xsl:for-each>
    ) VALUES (
                <xsl:for-each select="*">
                <xsl:if test="node()/name() != ' '">
                    <xsl:choose>
                    <xsl:when test="node() castable as xs:integer">
                    <xsl:value-of select="node()"/>
                    </xsl:when>
                    <xsl:otherwise>
                    '<xsl:value-of select="node()"/>'
                    </xsl:otherwise>
                    </xsl:choose>
                    <xsl:if test='position() != last()'>,
                    </xsl:if>
                </xsl:if>
                </xsl:for-each>
    );

    </xsl:if>

</xsl:template>
</xsl:stylesheet>
```

Con respecto a la hoja presentada en el apartado anterior, podemos observar diferencias notables. La tabla de estilos puede encontrarse en la ruta del repositorio "*stylesheets/SQL Generico 2.0.xsl*" o bajo el mismo nombre de archivo en la entrega.

Una de ellas es la anotación *"<xsl:output method="text">"*, esta anotación nos permite configurar cual será el resultado de la transformación aplicada. Los valores soportados por *method* pueden ser "*text*", "*xml*", "*html*" y "*name*"; en nuestro caso necesitamos usar "text" para indicar que queremos que acabe en texto plano, ya que luego al estar escrito con sintaxis SQL será *SQL Developer* el que lo interprete.

Otra de las diferencias es la presencia de una sentencia if ("*<xsl:if test="node()/name()" ¡= ' ' ">*") empleada para eliminar los nodos de elemento que no contienen ningún nodo de texto, ya que la sentencia SQL resultado para insertar objetos en una tabla debe listar en que columnas se encuentran los valores introducidos.

Continuando con los cambios introducidos, contamos con otra sentencia if ("*<xsl:if test='position() ¡= last()'>*") que determina si el nodo que el bucle *for-each* está recorriendo es el último. Esta sentencia se incluyó debido al problema a la hora de poner las comas cuando recorríamos los nodos de elemento que corresponden a las tablas. Para la condición asociada, empleamos las funciones "*position*" y "*last*" que nos permiten obtener la posición del bucle que se está interpretando y comprobar en cual posición se encuentra el intérprete.

Posteriormente definimos la parte de la sentencia SQL que indica los valores a insertar en las columnas definidas anteriormente. Es aquí donde podemos encontrar las sentencias *"<xsl:choose>"*, *"<xsl:when>"* y *"<xsl:otherwise>"*. Estas tres sentencias funcionan de manera similar a un "*if-else*" en cualquier otro lenguaje.

En el interior de estas sentencias definimos las condiciones para interpretar que clase de dato se encuentra en el nodo de texto. Por ello, hemos puesto como condición al primero que intente proyectar el dato como un número entero (*integer*) con la siguiente condición: "*node() castable as xs:integer*".

Si la condición definida se da, se transforma únicamente aquello contenido en el nodo de texto. Mediante el equivalente a *else*: *xsl:otherwise*; encontramos que los datos contenidos en el nodo de texto se les incluyen comillas ('*dato'*) para indicar que es de tipo *varchar* (Cadena de texto en sintaxis SQL).

Por último y no menos importante, hablaremos de la sentencia stylesheet. Para cambiar la versión de XSLT usada, así como definir nuevas dependencias hemos cambiado los siguientes parámetros:

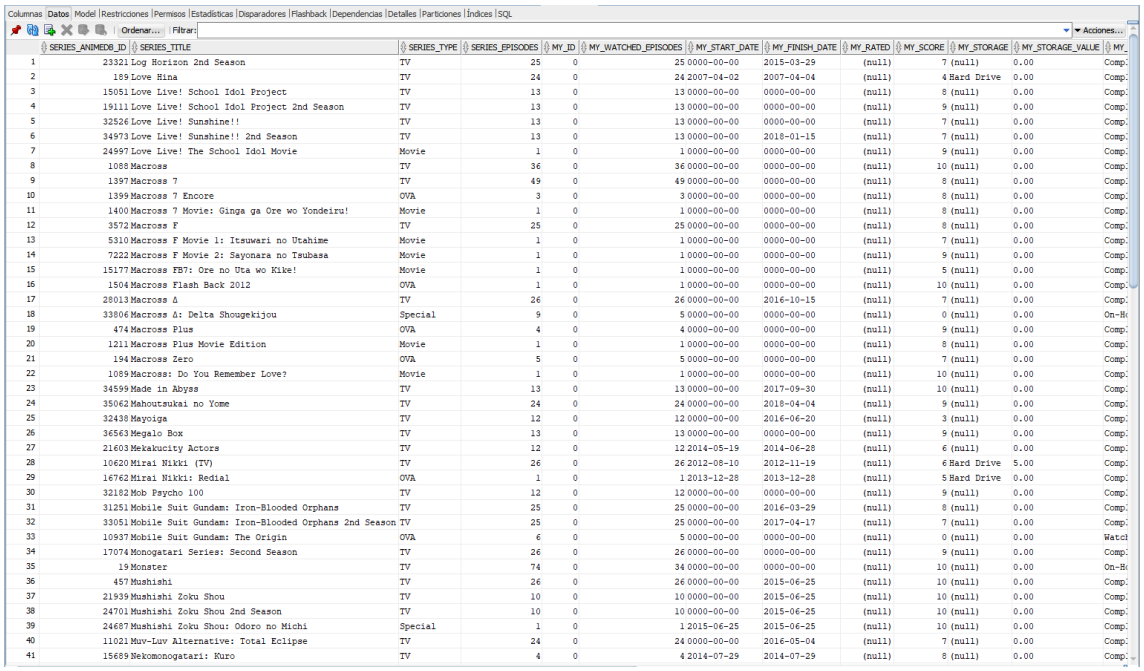
- "*version*" de 1.0 a 2.0
- Entre las librerías usadas, hemos solicitado la librería "*xs*" para los tipos de datos.

Habiendo analizado la hoja de estilos desarrollada, se ha procedido a realizar la transformación. En la siguiente captura podemos ver una porción del resultado:

```
INSERT INTO myinfo(user_id,user_name,user_export_type,user_total_anime,user_total_w
    ) VALUES (
                3256291,

                    'Acofmelt'
                    ,
                    1,
                    332,
                    6,
                    307,
                    9,
                    9,
                    1
    );


INSERT INTO anime(series_animedb_id,series_title,series_type,series_episodes,my_id,
    ) VALUES (
                35248,

                    '18if'
                    ,

                    'TV'
                    ,
                    13,
                    0,
                    5,

                    '0000-00-00'
                    ,

                    '0000-00-00'
                    ,
                    0,

                    '0.00'
                    ,

                    'Dropped'
                    ,

                    'No tenia tiempo pa verla, perdí el interés.'
                    ,
```

El resultado de esta transformación puede encontrarse en la ruta del repositorio "*results/Resultado SQL Generico 2.0.sql*" o entre los documentos de la entrega con el mismo nombre.

Tras esto, procedemos a introducir el resultado en la base de datos escogida. Notar que hemos creado las tablas de antemano.



Columnas | Datos | Model | Restricciones | Permisos | Estadísticas | Disparadores | Flashback | Dependencias | Detalles | Particiones | Índices | SQL

Ordenar... | Filtrar: | Acciones...

	SERIES_ANIMEDB_ID	SERIES_TITLE	SERIES_TYPE	SERIES_EPISODES	MY_ID	MY_WATCHED_EPISODES	MY_START_DATE	MY_FINISH_DATE	MY_RATED	MY_SCORE	MY_STORAGE	MY_STORAGE_VALUE	MY_
1	23321	Log Horizon 2nd Season	TV	25	0	25	0000-00-00	2015-03-29	(null)	7	(null)	0.00	Comp:
2	189	Love Hina	TV	24	0	24	2007-04-02	2007-04-04	(null)	4	Hard Drive	0.00	Comp:
3	15051	Love Live! School Idol Project	TV	13	0	13	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
4	19111	Love Live! School Idol Project 2nd Season	TV	13	0	13	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
5	32526	Love Live! Sunshine!!	TV	13	0	13	0000-00-00	0000-00-00	(null)	7	(null)	0.00	Comp:
6	34973	Love Live! Sunshine!! 2nd Season	TV	13	0	13	0000-00-00	2018-01-15	(null)	7	(null)	0.00	Comp:
7	24997	Love Live! The School Idol Movie	Movie	1	0	1	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
8	1088	Macross	TV	36	0	36	0000-00-00	0000-00-00	(null)	10	(null)	0.00	Comp:
9	1397	Macross 7	TV	49	0	49	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
10	1399	Macross 7 Encore	OVA	3	0	3	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
11	1400	Macross 7 Movie: Ginga ga Ore wo Yondeiru!	Movie	1	0	1	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
12	3572	Macross F	TV	25	0	25	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
13	5310	Macross F Movie 1: Itsuwari no Utahime	Movie	1	0	1	0000-00-00	0000-00-00	(null)	7	(null)	0.00	Comp:
14	7222	Macross F Movie 2: Sayonara no Tsubasa	Movie	1	0	1	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
15	15177	Macross FB7: Ore no Uta wo Kike!	Movie	1	0	1	0000-00-00	0000-00-00	(null)	5	(null)	0.00	Comp:
16	1504	Macross Flash Back 2012	OVA	1	0	1	0000-00-00	0000-00-00	(null)	10	(null)	0.00	Comp:
17	28013	Macross Δ	TV	26	0	26	0000-00-00	2016-10-15	(null)	7	(null)	0.00	Comp:
18	33806	Macross Δ: Delta Shougekijou	Special	9	0	5	0000-00-00	0000-00-00	(null)	0	(null)	0.00	On-H
19	474	Macross Plus	OVA	4	0	4	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
20	1211	Macross Plus Movie Edition	Movie	1	0	1	0000-00-00	0000-00-00	(null)	8	(null)	0.00	Comp:
21	194	Macross Zero	OVA	5	0	5	0000-00-00	0000-00-00	(null)	7	(null)	0.00	Comp:
22	1089	Macross: Do You Remember Love?	Movie	1	0	1	0000-00-00	0000-00-00	(null)	10	(null)	0.00	Comp:
23	34599	Made in Abyss	TV	13	0	13	0000-00-00	2017-09-30	(null)	10	(null)	0.00	Comp:
24	35062	Mahoutsukai no Yome	TV	24	0	24	0000-00-00	2018-04-04	(null)	9	(null)	0.00	Comp:
25	32438	Mayoiga	TV	12	0	12	0000-00-00	2016-06-20	(null)	3	(null)	0.00	Comp:
26	36563	Megalo Box	TV	13	0	13	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
27	21603	Mekakucity Actors	TV	12	0	12	2014-05-19	2014-06-28	(null)	6	(null)	0.00	Comp:
28	10620	Mirai Nikki (TV)	TV	26	0	26	2012-08-10	2012-11-19	(null)	6	Hard Drive	5.00	Comp:
29	16762	Mirai Nikki: Redial	OVA	1	0	1	2013-12-28	2013-12-28	(null)	5	Hard Drive	0.00	Comp:
30	32182	Mob Psycho 100	TV	12	0	12	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
31	31251	Mobile Suit Gundam: Iron-Blooded Orphans	TV	25	0	25	0000-00-00	2016-03-29	(null)	8	(null)	0.00	Comp:
32	33051	Mobile Suit Gundam: Iron-Blooded Orphans 2nd Season	TV	25	0	25	0000-00-00	2017-04-17	(null)	7	(null)	0.00	Comp:
33	10937	Mobile Suit Gundam: The Origin	OVA	6	0	5	0000-00-00	0000-00-00	(null)	0	(null)	0.00	Watch
34	17074	Monogatari Series: Second Season	TV	26	0	26	0000-00-00	0000-00-00	(null)	9	(null)	0.00	Comp:
35	19	Monster	TV	74	0	34	0000-00-00	0000-00-00	(null)	10	(null)	0.00	On-H
36	457	Mushishi	TV	26	0	26	0000-00-00	2015-06-25	(null)	10	(null)	0.00	Comp:
37	21939	Mushishi Zoku Shou	TV	10	0	10	0000-00-00	2015-06-25	(null)	10	(null)	0.00	Comp:
38	24701	Mushishi Zoku Shou 2nd Season	TV	10	0	10	0000-00-00	2015-06-25	(null)	10	(null)	0.00	Comp:
39	24687	Mushishi Zoku Shou: Odoro no Michi	Special	1	0	1	2015-06-25	2015-06-25	(null)	10	(null)	0.00	Comp:
40	11021	Muv-Luv Alternative: Total Eclipse	TV	24	0	24	0000-00-00	2016-05-04	(null)	7	(null)	0.00	Comp:
41	15689	Nekomonogatari: Kuro	TV	4	0	4	2014-07-29	2014-07-29	(null)	8	(null)	0.00	Comp:

Podemos observar en la captura que se han introducido correctamente los datos contenidos en el fichero xml, ignorándose las columnas que no contenían valores en sus nodos de texto.

3. Discusión

Mediante las plantillas que hemos elaborado a lo largo del proyecto, hemos observado ciertas ventajas e inconvenientes. En primer lugar, cabe destacar que puede llevarse a cabo, de forma sencilla, plantillas *XSLT* para documentos *XML* concretos con el fin de darles un formato *HTML*, así como para crear un script de *SQL* para introducir la información existente en una base de datos. A nuestro parecer, esta es una de las grandes ventajas de la creación de plantillas *XSLT*, pero por otra parte nos encontramos con mayores dificultades a la hora de crear una plantilla de *XSLT* genérica.

Mientras que una plantilla *XSLT* creada para un caso concreto tiene como única complejidad introducir el nombre de las variables que queremos obtener y el uso de condicionales para filtrar, crear una plantilla genérica que pueda ser usada para *XMLs* diferentes conlleva un análisis en profundidad y diversos problemas que hacen su elaboración más difícil.

En primer lugar, un *XML* puede poseer una cantidad de nodos anidados indefinida. Este suceso puede no conllevar una mayor dificultad si nuestro objetivo es mostrar la información en formato *HTML*, pero si nuestro objetivo es crear un script *SQL*, nos encontraremos con diversos inconvenientes que obstaculizarán la elaboración de una plantilla *XSLT* genérica. Sin ir más lejos, la creación de tablas para la base de datos supone un gran obstáculo, pues al recorrer el *XML* al completo, resulta complejo que nuestra plantilla diferencie las diversas tablas que debe crear y no intente crear más de una vez una tabla con el mismo nombre, lo cual generará un error.

Por otra parte, si suponemos que la creación de tablas se llevará a cabo independientemente por el propio usuario, entonces podremos generar mediante una plantilla *XSLT* un script de *SQL* que introduzca los diversos objetos con sus correspondientes atributos en las tablas creadas previamente, tal como hemos visto anteriormente en este mismo proyecto.

En cualquier caso, observamos que la creación de plantillas en *XSLT* conlleva un tiempo de aprendizaje minúsculo en comparación con los resultados obtenidos. La magnitud de trabajo que conllevaría insertar manualmente la información de un *XML* en una base de datos SQL podría ser extremadamente elevada, mientras que elaborar una plantilla específica *XSLT* para crear dicho script automáticamente supone un trabajo ínfimo en comparación, por lo que aún sin la creación de plantillas genéricas que nos sirvieran para la mayoría de casos, realizar una plantilla específica para la situación que queremos afrontar sigue siendo una sencilla tarea que nos ahorrará una extrema carga de trabajo en comparación.

4. Conclusiones

A lo largo de este proyecto hemos observado algunas de las utilidades que nos puede proporcionar las *transformaciones XSL* así como sus resultados. Desde la representación visual de los datos hasta la elaboración automática de *scripts SQL*, estas plantillas nos permiten darle el formato que nos convenga a toda la información que se encuentre en un *XML*.

Además, la sencillez del lenguaje permite elaborar rápidamente las plantillas que necesitemos según el objetivo que estemos intentado conseguir, lo cual nos permitirá aumentar nuestra productividad y realizar tareas que podrían conllevar días en cuestión de minutos.

Aunque la creación de plantillas genéricas que puedan usarse en diferentes casos supone un gran aumento de la dificultad, la elaboración de plantillas específicas para el caso que queramos tratar suele suponer tan poca dificultad, que en muchas ocasiones es posible que nos sea más conveniente elaborar las plantillas específicas que necesitemos antes que una genérica para todas ellas.

Como mejoras propuestas a este trabajo, se puede experimentar con la transformación a otros formatos de amplio uso como *Json* o la elaboración de hojas de estilo aún más complejas que permitan procesar cualquier tipo de fichero *XML* de entrada.

5. Referencias

(s.f.). Obtenido de XSLTTest: https://xslttest.appspot.com/

Andagualh, P. (s.f.). Obtenido de Github: https://github.com/Andagualh/XSLT-AnimeListFormatter

Orsini, L. (30 de 5 de 2018). Obtenido de Forbes: https://www.forbes.com/sites/laurenorsini/2018/05/30/myanimelist-passes-third-day-of-unexpected-downtime/?sh=13714a477e9e

Peters, M. (30 de 5 de 2018). Obtenido de Comicbook: https://comicbook.com/anime/news/myanimelist-down-issue-gone-security-problem/

Roberts, I. (29 de 8 de 2013). Obtenido de StackOverFlow: https://stackoverflow.com/questions/18513376/xslt-multiple-templates-with-same-match

Stein, B. (s.f.). Obtenido de Github: https://github.com/bramstein/xsltjson

Varios, a. (29 de 4 de 2021). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Extensible_Markup_Language

Varios, a. (9 de 2 de 2021). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Extensible_Stylesheet_Language

Varios, a. (9 de 2 de 2021). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations

W3School. (s.f.). Obtenido de W3School: https://www.w3schools.com/xml/xsl_intro.asp