Advanced Databases Summative

Bradley Mackey

for 15th March 2019

1 Tree Structure

See the attached document for the tree diagram of `books.xml`. This is available on the **final page** of this report. Any oddities in the layout are only a result of a lack of space.

2 Querying

All query files were posted as part of this submission.

2.1 XPath

2.1.1 Find all Mary Poppins books which were published before WWII.

1. *Query*:

```
<books>{
    doc("books.xml")
    //book[@publicationDate<"1939"]
    [title[distinct-values(contains(text(),"Mary Poppins"))]]
}</books>
```

2. *Interpretation*: This query first uses the Restricted Kleene Closure to find every occurrence of `<book>` in the XML structure. This ensures that we find all books at all levels of the database (if the database were to be grown in the future or if books were to be located under differently named nodes to what is currently in place). We then use a tag predicate of `[@publicationDate<"1939"]` to filter out all books that were NOT released before 1939. For all the books returned by this, we then check if they have a `title` property. If they do, we use a nested predicate to check the title of the book. If the book title contains the string `"Mary Poppins"` (not including these quotation marks), the full book node will be returned. Only distinct books titles will be included in our results, as we make use of the `distinct-values` function to ensure that each Mary Poppins book is only included in the result once.

3. *Limitations*: this query will only find Mary Poppins books if her name appears directly in the title. If such a Mary Poppins book existed that did not feature her name in the title, it would not be returned by our query. It is not sufficient just to check the author or illustrator as they may have been involved in creating other books (not Mary Poppins). Additionally, WWII started in September of 1939 and—as we have no way to distinguish

the month of publication—it may be the case that a book published in 1939 was actually published before WWII. We have no way of telling this, we just ignore all books published in 1939 to be sure.

4. *Output*:

```
<books>
  <book publicationDate="1934">
    <title>Mary Poppins</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
  <book publicationDate="1935">
    <title>Mary Poppins Comes Back</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
</books>
```

2.1.2 Find all movies that have academy nominations.

1. *Query*:

```
<movies>{
    doc("media.xml")
    //movie[normalize-space(academyNominations/text()) > "0"]
}</movies>
```

2. *Interpretation*: This query first uses the Restricted Kleene Closure to find every occurrence of `<movie>` in the XML structure (we assume "movie" refers to the `<movie>` tag rather than the contents of the `@type` attribute on each `<movie>`). This ensures that we find all movies at all levels of the database (if the database were to be grown in the future or if movies were to be placed elsewhere in a large database). We then use a tag predicate of `academyNominations`, filtering out all movies that do not include an `<academyNominations>` node. We access the text stored at this node using the XPath `text()` function. As the XML files provided have whitespace surrounding the text in each node, we use the `normalize-space` function to remove this whitespace, as to not retrieve inaccurate results. We then ensure that the number of nominations is greater than 0, as it could be possible a movie could have an `<academyNominations>` node with 0 nominations. Movie nodes filtered out by this predicate are not included in our results.

3. *Limitations*: this query relies on the fact that `academyNominations` are stored as an integer inside the `<academyNominations>` node. If the nominations were instead listed out (as `<nominator>` nodes for example), the query would not be able to identify these. Otherwise, if all movies follow this style of listing nominations, there are no further limitations. If one of the other entities in the `media.xml` file were to receive an `academyNominations` (such as `broadcast` or `theater`), these would not be found, as we only search for `movie` entities. Although, as this is what the question specifically asks for, this is less of an issue.

4. *Output*:

```
<movies>
  <movie year="1964" type="film" language="En">
    <title>Mary Poppins</title>
    <actress>Julie Andrews</actress>
    <actor>Dick Van Dyke</actor>
    <actor>David Tomlison</actor>
    <actress>Glynis Johns</actress>
    <producer>Walt Disney</producer>
    <director>Robert Stevenson</director>
    <academyNominations>13</academyNominations>
  </movie>
  <movie year="2013" type="film" language="En">
    <title>Saving Mr. Banks</title>
    <actress>Emma Thompson</actress>
    <actor>Tom Hanks</actor>
    <academyNominations>1</academyNominations>
  </movie>
</movies>
```

2.2 XQuery

2.2.1 How many books have there been published about Mary Poppins?

1. *Query*:

```
let $b := distinct-values(doc("books.xml")//book[title[contains(text(),"Mary Poppins")]])
return <result>{count($b)}</result>
```

2. *Interpretation*: we start by using the `let` keyword to bind to all results of a query at once, allowing us to perform aggregate operations on the results of the query. We open the `"books.xml"` file such that we can query it. Using a similar XPath style query as question 1, we query the document for all `book` nodes using the Restricted Kleene Closure. This will allow us to find all possible `<book>` nodes at any point in the database, even if the database were to grow in size in the future. We then use a predicate to assert that each returned `book` has a `title` property. Then, we use a nested predicate to evaluate the text of the title, using the `text()` function. We assert that the title must contain the string `"Mary Poppins"`, such that all books about Mary Poppins are returned. We use the `distinct-values` function to ensure that if there were to be duplicate books found in the database with the same book title, this will only be counted once. Then, in the `return` block, we perform an aggregate `count` of all the results, returning this in a `<result>` tag. This formatting of the answer would help a parser interpret this result potentially, making it clear it is the result of the query.

3. *Limitations*: Similar to question 1, this will only count books where `"Mary Poppins"` appears in the title directly. If there is a book that is actually about Mary Poppins but does not include her name in the title, it will not be included in the book count.

4. *Output*:

```
<result>8</result>
```

5. *XPath*: As XPath and XQuery share the same functions library, we are able to construct an XPath query to accomplish the same functionality. We use the same path component with exactly the same predicates as the XQuery query. We then wrap this within the `count` function that simply counts the number of returned results. As the results returned by an XPath operation are akin to what is bound by a `let` statement in XQuery, this operation is seamless:

```
<result>{
    count(
    distinct-values(
    doc("books.xml")//book[title[contains(text(),"Mary Poppins")]]
    ))
}</result>
```

2.2.2 List the books that have been published before the second Mary Poppins movie appeared.

1. *Query*:

```
let $yrs := (
    for $mov in doc("media.xml")//movie
    where distinct-values($mov[title[contains(text(),"Mary Poppins")]])
    order by number($mov/@year)
    return $mov
)
let $yr2 := number($yrs[2]/@year)
return <books>
{
for $b in doc ("books.xml")//book
where $b[number(@publicationDate) < $yr2]
return $b
}
</books>
```

2. *Interpretation*: we firstly need to find what year the second Mary Poppins movie appeared. Therefore, we perform a nested query to return a list of Mary Poppins movies, ordered by the year that they were released (making sure that there are no duplicated movie titles in this list by using the `distinct-values` function). We search for movies using the Restricted Kleene Closure to ensure that we are able to find all movies if more were to be added to different places in the dataset in the future. We directly index into the returned list using `$yrs[2]/@year` to get the year that the second movie was released. This ordering and indexing is a nice interface to easily change this value in the future if the database required it—we simply just change the number of the index to get books released before that specific movie. We then fetch all books from the books database, using a `where` predicate to only return books that were released before this second movie was

released by comparing the `@publicationDate` to the value of `$yr2`. We then return each result that matches this predicate as the successful query cases, wrapped in a `<books>` result object to make this result easier to parse and interpret as a list of results. This is accomplished using a nested query.

3. *Limitations*: this technique suffers from the same limitations as the previous queries, as movies without `"Mary Poppins"` in the title will not be considered Mary Poppins movies, as these are what is matched against. Additionally, in order for us to find the second released movie, we sort through all movies with Mary Poppins in the title first. If the database contained many movies with `"Mary Poppins"` in the title, this sorting could take a non-trivial amount of time and impact database performance (as sorting is at least an $\mathcal{O}(n \log n)$ operation).

4. *Output*:

```
<books>
  <book publicationDate="1934">
    <title>Mary Poppins</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
  <book publicationDate="1935">
    <title>Mary Poppins Comes Back</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
  <book publicationDate="1943">
    <title>Mary Poppins Opens the Door</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
  <book publicationDate="1952">
    <title>Mary Poppins in the Park</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
  <book publicationDate="1962">
    <title>Mary Poppins From A to Z</title>
    <author>P. L. Travers</author>
    <illustrator>Mary Shepard</illustrator>
  </book>
</books>
```

5. *XPath*: If we assume both datasets are present in the same file (where the `<books>` and `<media>` nodes are both present within a `<root>` node, which we will call `combined.xml`), we can construct a working query—otherwise this would not be possible, as XPath is not able to access more than one file at once. We must assume that the nodes in the `<books>` and `<media>` nodes are already ordered by date, which is the major limitation of this

technique compared to what we can achieve with XQuery. We return all books where the numeric value of the `publicationDate` is less than the release year of the second movie that appears in the returned list of movies about Mary Poppins. This query is not as extensible if the size of the database were to be increased.

```
<books>{
    doc("combined.xml")//book[
    number(@publicationDate/string())
    <
    number(
        ancestor::root
        //movie[title[distinct-values(contains(text(),"Mary Poppins"))]][2]
        /@year
    )
]
}</books>
```

2.2.3 How much later has the most recent Mary Poppins movie appeared, compared to the second book about Mary Poppins?

1. *Query*:

```
let $byrs := (
    for $b in doc("books.xml")//book
    where distinct-values($b[title[contains(text(),"Mary Poppins")]])
    order by number($b/@publicationDate)
    return $b
)
let $byr := number($byrs[2]/@publicationDate)
let $myrs := (
    for $mov in doc("media.xml")//movie
    where distinct-values($mov[title[contains(text(),"Mary Poppins")]])
    order by number($mov/@year) descending
    return $mov
)
let $myr := number($myrs[1]/@year)
let $diff := sum ( ($myr, -$byr) )
return <result>{$diff} years</result>
```

2. *Interpretation*: similar to the previous question, we firstly order all books by their publication year so we know that index positions directly correspond to the year they were released. We then get the `publicationDate` of the second book to be released indexing and converting to a number by performing `number($byrs[2]/@publicationDate)`. We perform a similar operation on the movies, except ordering these descending such that we can easily access the most recent movie, ordering by the `year` property of each `movie`. We then get the year of the most recent movie and convert this to a number by doing `number($myrs[1]/@year)`. For each of these, we ensure that each title is distinct, such that we do not count the same entry twice if the database were to grow and include duplicated values. Then we calculate the difference between these two years using the XQuery library function `sum`, which is able to add a list of numbers, returning a numeric result. The

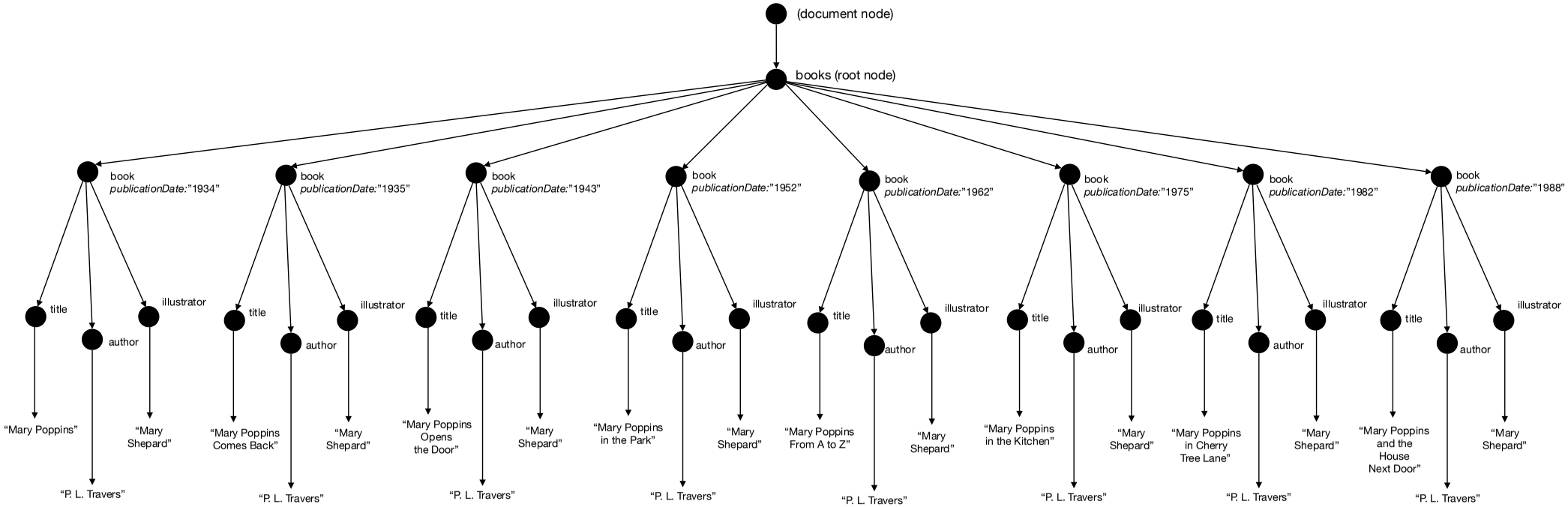
equation performed is the equivalent of $year_{movie} - year_{book}$. The result is then returned in a `result` node.

3. *Limitations*: Again, similar to the prior question, as this technique requires sorting it may not scale well as the database grows. As this sorts **both** the movie and books dataset, this will scale even worse than the prior question. Similar to the prior questions as well, this identifies Mary Poppins movies and books by seeing if her name is contained in the title. If it is not, it will not be successfully identified as a Mary Poppins movie or book.

4. *Output*:

```
<result>83 years</result>
```

5. *XPath*: This is not a query that is possible for XPath to handle. This is due to the fact there are dependencies on accessing 2 independent lists of data and then the performing of an operation of the result of the two independent lists. As XPath is only able to access a single list of data (and possible compare each value in this list with a single element from another list), this is not able to be done.


(document node)
books (root node)
book publicationDate:"1934"
book publicationDate:"1935"
book publicationDate:"1943"
book publicationDate:"1952"
book publicationDate:"1962"
book publicationDate:"1975"
book publicationDate:"1982"
book publicationDate:"1988"
title
author
illustrator
"Mary Poppins"
"P. L. Travers"
"Mary Shepard"
"Mary Poppins Comes Back"
"Mary Poppins Opens the Door"
"Mary Poppins in the Park"
"Mary Poppins From A to Z"
"Mary Poppins in the Kitchen"
"Mary Poppins in Cherry Tree Lane"
"Mary Poppins and the House Next Door"