The benchmarking XQuery and OOXQuery for ease of use and performance

by

Ying Wei

A report submitted to the graduate faculty

of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Dr. Gadia Sashi, Major Professor

Dr. Ying Cai

Dr. Simanta Mitra

Iowa State University

Ames, Iowa

2018

Abstract

XML – the extensible markup language – is a versatile platform for representation of information. Using the type mechanisms available within XML, we have proposed OOXML, an object-oriented dialect for XML. XML elements can be realized as objects by enhancing them to support object-identities, object hierarchy, and references. As OOXML documents are legal XML documents, they are amenable for use in XML processing technologies such as XQuery, XSLT, DOM/API, etc. In addition to the usual syntax the users can benefit from dotted expressions of object-oriented systems to access properties of super and referenced objects, we have implemented a preprocessor for OOXQuery – a dialect of XQuery – that translates an OOXQuery query into XQuery query. Although the translated query is cryptic for users, it can be processed in an XQuery engine. In this paper we consider benchmarking XQuery and OOXQuery for ease of use and runtime performance. Due to dotted expressions, OOXQuery is generally more natural than XQuery. For many types of queries, the performance of OOXQuery seems better. Therefore, we find that OOXQuery is an interesting alternative to XQuery.

The object hierarchy is available in two flavors: hierarchy by reference and hierarchy by value for which we informally use suffixes "-R" and "-V", respectively, when necessary. Within the two options for hierarchy, OOXQuery-R seems more natural than OOXQuery-V and the performance results are mixed.

1. Introduction

We have implemented OOXML, an object-oriented dialect of XML [1], where users can realize XML elements as objects by enhancing them to support object-identities, object hierarchy, and references. OOXML documents are legal XML documents, hence they can be deployed using XML processing technologies such as XQuery, XSLT, DOM/API, etc. In addition to the usual syntax the users can benefit from dotted expressions of object-oriented systems to access properties of super and referenced objects, we have implemented a preprocessor for OOXQuery – a dialect of XQuery [1] that translates an OOXQuery query into XQuery query. Although the translated query is cryptic for users, it can be processed in an XQuery engine. In this paper we consider benchmarking XQuery and OOXQuery for ease of use and runtime performance.

The main difference between XQuery and OOXQuery is that whereas in the former a user thinks in terms of joins in the latter in terms of dot expressions. We find that generally OOXQuery seems more natural for users. However, the performance results are mixed. Although in some cases XQuery outperforms OOXQuery, there are sufficiently many cases where OOXQuery does better. This means OOXML and OOXQuery are worthy of consideration as alternative to XML for users who prefer to develop applications in object-oriented style.

The object hierarchy is available in two flavors: hierarchy by reference and hierarchy by value for which we informally use suffixes "-R" and "-V", respectively, when necessary. In OOXML-R the super-object properties of objects are physically left at super-object level and simply referenced when needed. In OOXML-V the super-object properties of an object are materialized with the object – by value. OOXQuery-R and OOXQuery-V are also compared. Here we find

that OOXQuery-R seems more natural than OOXQuery-V and the performance results for the two are mixed.

We carefully choose our usecase for benchmarking for comparing ease of use and performance. The usecase consists of XML and OOXML schemas, the XML and OOXML documents, and queries – first expressed in English and then XQuery and OOXQuery. The schema is first introduced in terms of Entity Relationship Model [3]. Then XML and OOXML schemas are introduced. A merit of the usecase is that the elements in XML correspond rather directly with objects in OOXML. This makes the comparison between XQuery and OOXQuery as even handed as possible. The main difference for the user is that in XQuery a user thinks in terms of joins and in OOXQuery in terms of dotted expressions. Three categories of queries are considered. The first category, considered in Section [3.1], is meant to compare joins in XQuery and dotted expressions in OOXQuery. The second category starts by noticing that, whereas in XML there are six XML documents, in OOXQuery their information content has to be centralized in a single OOXML document that is much larger. Due to largeness, the comparison between XQuery and OOXQuery a bit more attention. Some evidence is given to support the argument that clever implementations of OOXQuery engine can help alleviate this situation. In the third category, we compare OOXQuery-R and OOXQuery-V.

Our experimentation is conducted in an interesting style. We use Cyclone Database Implementation Workbench (CyDIW) [4], a tool that allows considerable amount of automation. We note that OOXQuery is quite powerful and it is used as a utility to help us transform OOXML-R datasets into XML as well as OOXML-V datasets. CyDIW has built-in facility for performance benchmarking. Large chunks of the experiments are executed at single clicks of a button in CyDIW.

The rest of this paper is organized as follows. In Section [2] we introduce our use case for benchmarking with datasets of varying sizes. In Section [3] we describe the experiment using queries in three categories mentioned above. In Section [4] we conclude the paper. Several appendixes are included to describe the style followed by our experiments.

2. Usecase setup

We introduce our usecase for benchmarking via entity relationship model (ERM). This is a good practice in general, but in our context it is even more appropriate as among all data models, because object-oriented database models seem closest to ERM. The closeness should not be surprising as the core of ODMG proposal [5,6] as a standard for object-oriented databases seems to be derived directly from ERM. Whereas relationships in ERM are seen as undirected edges between entities, in ODMG proposal a relationship is broken into two directed edges. A reference is easily traversed using a dotted expression at syntax level in OQL-like query language. Object hierarchy in object oriented systems seems a direct counterpart of the isa-based hierarchy in ERM. Dotted expressions also help fetch properties of an object in the hierarchy. The counterpart of dotted expressions in the relational model are joins. This represents a major difference between XQuery and OOXQuery. A comparison of usability and performance of joins in XQuery and dotted expressions in OOXQuery are undertaken in Section 3.1.

In order to accomplish benchmarking we have chosen an interesting usecase, UniversityDB, that serves us rather well. Figure 1 shows the schema of UniversityDB as an entity relationship diagram. There are six entity types: Person, Instructor, Student, Course, Offering, and Enrollment. An instructor as well a student is a person. A student has a mentor who is an instructor. In academia, the concept of prerequisites of courses can be quite complex, but in our usecase we simply assume that a course can have other courses as prerequisites. Offering brings participation of courses and instructors together. Section is an attribute of Offering. A course can be offered via multiple sections. Finally, enrollment brings participation of offerings and students together. The grade of student is recorded in enrollment of a course.

UniversityDB as a usecase works well for OOXQuery as well as XQuery. An interesting feature of the use case is that the database schema in both cases include the same entity sets: Person, Instructor, Student, Course, Offering, and Enrollment. The fact that the entities in the two databases correspond directly with each other is helpful in a direct comparison of usability as well as performance. As mentioned before, navigation in OOXQuery is based on dot expressions whereas in XQuery it is natural-join based.

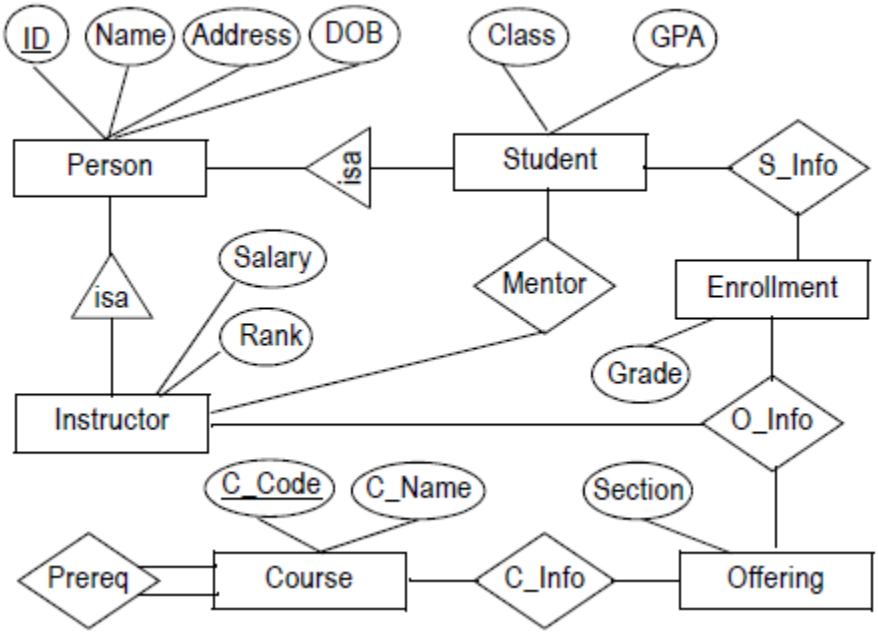

Figure 1. Schema for the University OOXML document

2.1. The schemas for the usecase

Based upon the ERM schema of Figure 1, the OOXML schema is shown in Figure 2. Corresponding to each of the six entity types we have six object types. Every object has an object identity: in addition to their structure, objects of the same type have identity that is uniformly typed. The two instances of isa are also incorporated so that an Instructor is a Person and Student is also a person. In the ERM we have 5 relationships. Each of them is converted into two directional references. For example, a student has a Mentor that is an instructor; on the other

hand an Instructor is Mentoring several students. In XML these can be captured using types IDREF and IDREFS, respectively. The directional relationships allow uninhibited navigation in the OOXML document as a graph using dotted expressions – hopping from node (entity) to node along edges (relationships).

WE caution the reader that Figure 2 does not capture the difference between schemas of the reference-based and value-based hierarchies. The differences show up in the XML code of the schemas as well as instance level documents. In OOXQuery-R as well as OOXQuery-V a user can use dots to access properties of super objects. In OOXQuery-V the user has an option of using "/" to access the super-object properties as they are materialized as children within an object. Thus from a syntax point of view there is not much difference between OOXQuery-R and OOXQuery-V. But the performance of queries show some differences between the two versions. This issue is investigated in Section 3.3.

Here, it is a good idea to clarify that OOXML documents (-R as well as -V) are legal XML documents and they are amenable to direct query using plain XQuery: instead of dotted expressions, one uses cryptic expressions involving '/" and id(.) function. But the dotted expressions in OOXQuery are much easier to use.

Next we consider the usecase for XQuery where the data consists of six XML files: Person.xml, Instructor.xml, Student.xml, Course.xml, Offering.xml, and Enrollment.xml. Their schemas are counterpart of the six entity types in Figure 2. Each attribute in ERM directly gives rise to a child element in corresponding XML file. (Examples of these are Name, DOB, Class, Rank, and Grade.) In addition, in XML files the attributes that mimic inter entity navigation implicitly supported by the two isa-based hierarchy and several binary relationships have to be included. The following shows the schemas of the six XML documents.

- Person (ID, Name, Address, DOB)
- Faculty (FacultyID, Rank, Salary)
- Student (StudentID, MentorID, Classification, GPA, CreditHours)
- Course (CourseCode, CourseName, PreReq)
- Offering (CourseCode, SectionNo, InstructorID)
- Enrollment (CourseCode, SectionNo, StudentID, Grade)

2.2. Creation of instance datasets

For XQuery, we store the dataset in six xml files. However, the situation is different for the OOXML document that is used by OOXQuery. In the current state of XML technology, all references can be resolved only within the same file. Therefore, for OOXQuery, the whole dataset would be stored in a single OOXML file. As OOXML-R and OOXML-V differ in the way properties of super objects are incorporated, there will be single centralized document corresponding to each version. One consequence of centralization is that in XQuery one deals with small XML documents but OOXQuery has to deal with a single, larger document. Because of this for the same query the performance of expressions in XQuery and OOXQuery can be different even though in OOXQuery much of the larger document is not relevant to the query. We consider this issue in Section 3.2.

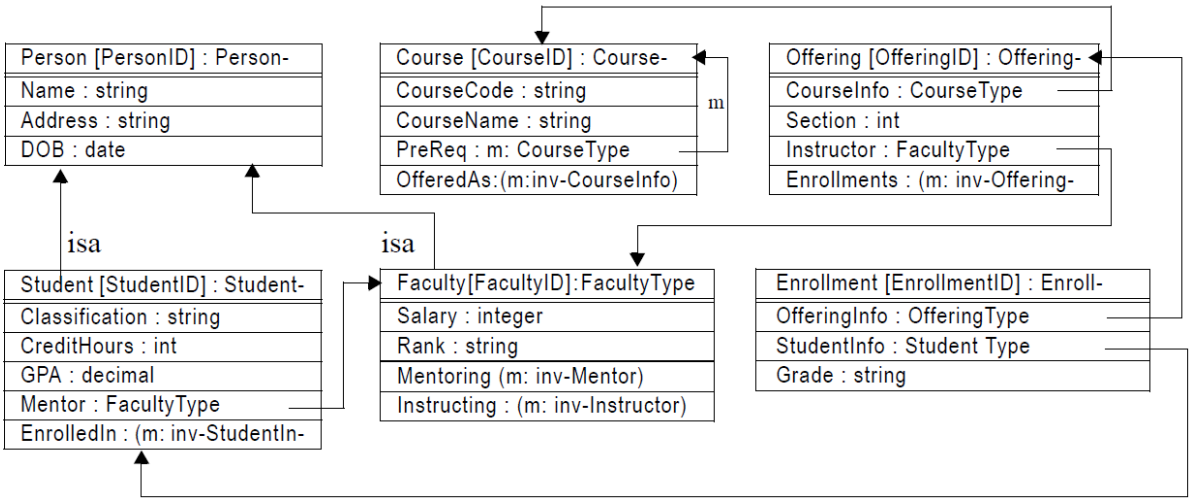

Figure 2. Schema for the University OOXML document

For benchmarking we consider datasets in 11 different sizes. The size of OOXML-R document ranges from 124 KB to 136 MB. First, a 124 KB OOXML-R document with six types of objects is manually populated. This is an OOXML document where the hierarchy is reference based. The number of objects in this database are successively doubled 10 times by a program written in DOM API. Each of these datasets is then converted into OOXML-V documents by using an OOXQuery query as a utility to conduct the experiment. For XQuery, an OOXML-R document is also converted into six XML documents by using an OOXQuery query. The datasets of different sizes are stored in 11 different folders in UniversityDB. The folders are named Size00, Size01, …, Size10. Each folder contains an OOXML-R document, one OOXML-V, and six XML documents, 8 different documents. The names of these documents does not change from one folder to another but it is the folder name that indicated the relative size. In preparation to address the size issue in Section 3.2, we will include two auxiliary OOXML-R documents in each size folder. The schema files of the six XML documents, the OOXML-R, and OOXML-V documents do not depend upon the size and they are stored centrally in the root of UniversityDB folder. The two auxiliary documents mentioned above will need their own schemas. These auxiliary schemas will also be stored in the root of UniversityDB folder. Thus the UniversityDB folder will have 10 schema documents for 6 to validate XML documents, 2 to validate OOXML-R and OOXML-V documents, and 2 to validate the two auxiliary OOXML-R documents.

The conversions of the datasets from OOXML-R to XML, OOXML-V, and the auxiliary OOXML-R documents is done in CyDIW that allows variables and for loops to eliminate duplication of code from one size to another. The entire conversion is done at a single click of a button in CyDIW. Although, not relevant to this paper, datasets for Neo4j database as well as MySQL can also generated in a similar manner using OOXQuery. Many details are covered in appendices in this paper. It is interesting that OOXQuery, which is the object of study in this case is also used as an infrastructural tool to help us carryout our experiments.

For some queries it is not necessary to use dataset of all eleven sizes for benchmarking because in almost all cases using seven of these suffice to reveal performance differences between XQuery and OOXQuery.

2.3. The system environment for testing

The experiment was performed on a PC with Intel Core i7-4500U CPU at 1.80GHz and with 8 GB memory. The system type is 64-bit operating system. CyDIW was installed on the PC. To generate graphical reports, the environment R has also been installed on the PC. A popular commercially available XQuery engine has been installed and configured to work directly from CyDIW. The performance of CyDIW is not a matter of concern as it acts only a facilitator in carrying out experiments without adding any significant overheads. Besides, every performance measurement is done as an small isolated task.

3. Experiments and results

Now we consider benchmarking XQuery and OOXQuery for ease of use and performance based upon a suite of queries. As mentioned earlier, we consider queries to address three issues: (i) joins in XQuery vs. dotted expressions in OOXQuery; (ii) the affect of centralization of data in a single large OOXML document for OOXQuery; and (iii) comparison between inheritance by reference and inheritance by value.

Because of complex nature of caching in hardware, the query execution time is not stable, specially in initial runs. In order to get cleaner benchmarks, caches are first "warmed up" by executing a query three times without recording the execution times. Then a query is executed three more times and the execution times are recorded and then averaged. CyDIW allows XML based logging to mimic the structure of the experiments. Some details of this are shown in Appendix D.)

It is to be kept in mind that although our benchmarking is done on a commercially available XQuery engine, it may not create the necessary access methods leading to best possible performance. So the performance reports are not to be takes as a final word as query engines undergo improvements and acquire greater schema awareness. Our main focus is on a relative comparison between XQuery and OOXQuery in an effort to investigate if performance of OOXQuery is reasonable. Based upon the results we can conservatively state that generally OOXQuery seems to be easier to use and that there are no alarming issues with its performance when compared to XQuery. We have not considered update operations in this paper.

3.1. Joins in XQuery vs. a dotted expression in OOXQuery

We consider the following query: *List name and rank of Faculty whose salary is at least 10000*. The XQuery and OOXQuery expressions are shown below. Based upon the time taken for the datasets of varying sizes the performance is shown in Figure 3.

XQuery

```
for $i in doc("…/Faculty.xml")//Faculty
for $p in doc("…/Person.xml")//Person
where $i/Salary >= 100000 and $i/FacultyID = $p/ID
return <E> {$p/Name, $i/Rank} </E>
```

OOXQuery

```
for $f in doc("…/UnivOO-R.xml")//Faculty
where $f/Salary>=100000
return <E> {$f.Name, $f.Rank} </E>
```

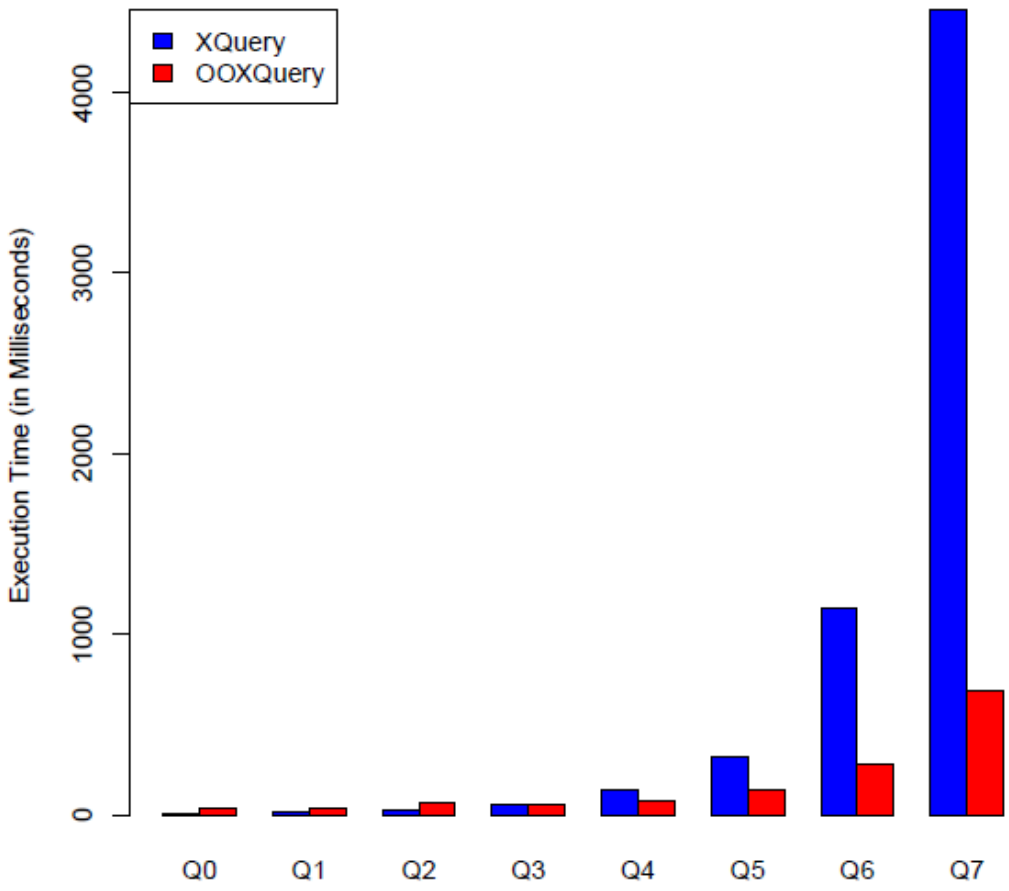

Figure 3. A 2-way XQuery join

For this query, the OOXQuery expression seems simpler than that in XQuery. In this case OOXQuery seems to perform better than XQuery. Both queries are executed on the same engine. (This will be true of all queries to follow.) Perhaps the example helps us to start to question any inhibitions one might have about performance of OOXQuery.

The internal translation of an OOXQuery into XQuery

The XQuery query is executed directly. But the OOXQuery query is first preprocessed to obtain an equivalent XQuery query before execution. The time for preprocessing is not significant. Just for the sake of illustration, in this case the translation is shown below. The translated query yields cryptic phrases using the id(.) function in XML and path expressions. The translational, execution, and benchmarking are preformed in CyDIW.

```
for $f in doc("C:/newDataSet/UniOO_R/UniOOR1.xml")//Faculty
where $f/Salary >= 100000
return <E> {$f/id($f/ISA/@oID)/Name , $f/Rank} </E> } </Item>
```

Now we consider a query that requires a 3-way join in XQuery: *List Student's name and their mentors' name*. The corresponding XQuery and OOXQuery expressions are shown below. Their performance is shown in Figure 4.

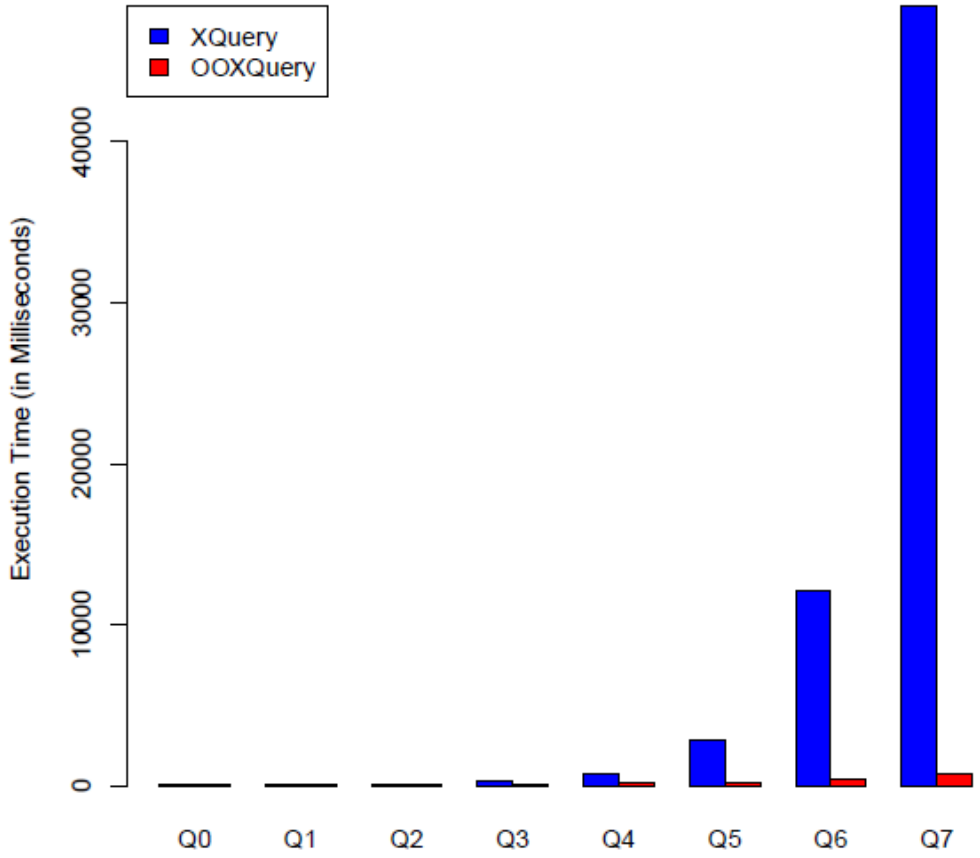

Figure 4. A 3-way XQuery join

XQuery

```
for $s in doc("…/Student.xml")//Student
for $p in doc("…/Person.xml")//Person[ID=$s/StudentID]
for $p1 in doc("…/Person.xml")//Person[ID=$s/MentorID]
return <E> {$p/Name, $p1/Name} </E>
```

OOXQuery

```
for $s in doc("…/UniOO-R.xml")//Student
return <E> {$s.Name, $s.Mentor.Name} </E>
```

Here we find that OOXQuery is simpler. Compared with the 2-way join above, for this query the performance of OOXQuery is seen to be significantly better than XQuery.

3.2. The issue of the dataset sizes in XML vs. OOXML

In order to articulate this issue we consider the query: *list courses and their prerequisite*. The XQuery and OOXQuery expressions for these are shown below. The performance is reported in Figure 5.

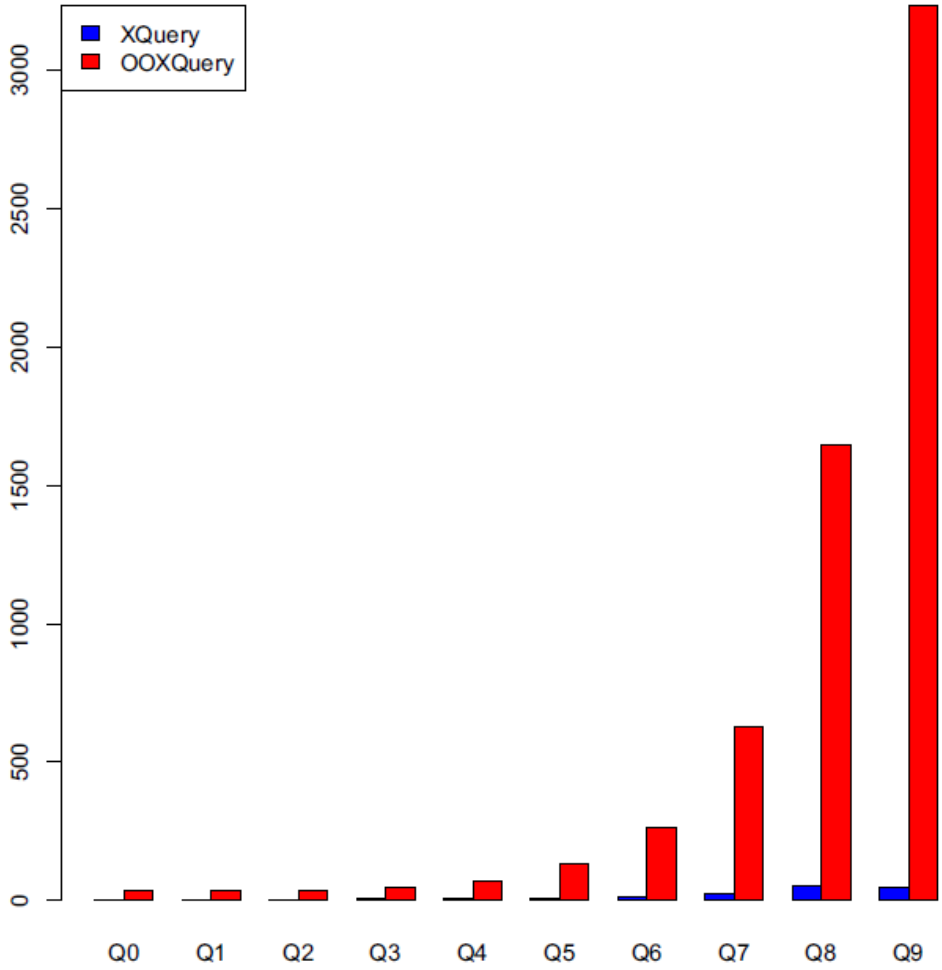

Figure 5. Prerequisites of courses

XQuery

```
for $c in doc("…/Course.xml")//Course
return <E> {$c/CourseCode, $c/PreReq} </E>
```

OOXQuery

```
for $c in doc("…/UnivOO-R.xml")//Course
return <E> {$c.CourseCode, $c.PreReqs.CourseCode} </E>
```

The two expressions seem equally easy in this case. But it turns out that the performance of OOXQuery in this case is much worse than XQuery. This may seem surprising at first. In order

to understand the underlying reason we look a bit deeply into how the data is organized in XML vs. OOXML.

The query only involves information within courses. XQuery needs Course.xml file (X Size07 dataset). Even though the OOXQuery only depends upon Course objects, it requires UnivOOR.xml file that contains all the six types of objects. XML requires that a reference originating in an XML document is resolved within the same document. Therefore, the OOXML dataset has to include all the six types of objects and hence much larger in size (X for Size07 dataset). We may suspect that the bad performance of OOXQuery in this case is due to the inflation in the size of the dataset. Therefore we consider altering the OOXML dataset. We consider two strategies.

• We place all the Course objects at the beginning in UnivOO-R.xml. This requires a new modified schema to be created for data validation. However, the performance does not improve.
• We remove all non-Course objects from UnivOO-R.xml. In this case too, the schema has to be redesigned as well.

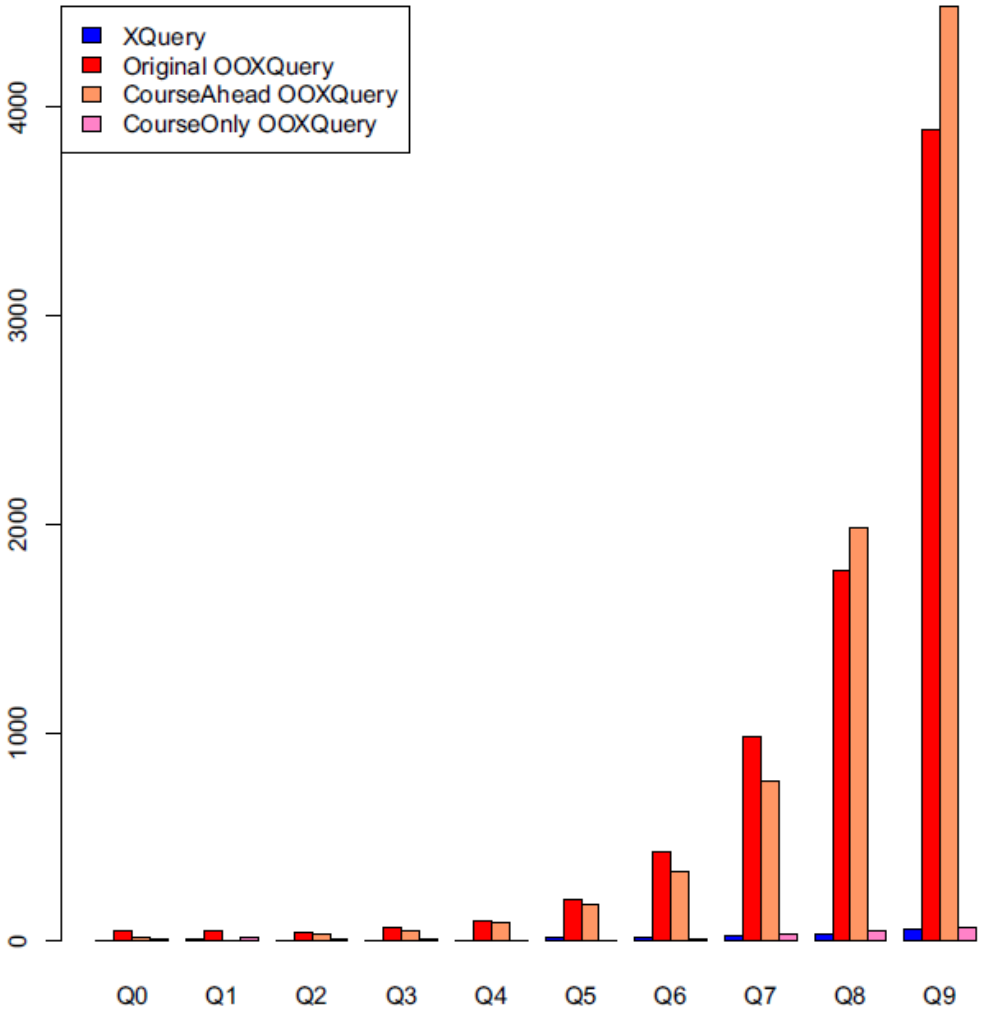

Figure 6. Scenarios for prerequisites of courses

Figure 6 reports the performance for various scenarios: (i) performance of XQuery, (ii) performance of OOXQuery without any changes in the OOXML dataset, (iii) performance of OOXQuery when the course objects are moved ahead in the OOXML document, and (iv) performance of OOXQuery when all but the course objects are eliminated. We find that the performances in (i) and (iv) are quite comparable. This validates our suspicion that bad performance in (ii) is not due to OOXQuery but the inability of the XQuery engine to isolate Course objects for efficient execution of the query. The query engine needs to be more schema aware and automatically take care of such isolation by including appropriate access methods.

3.2.1. Query that involves a 3-way join in XQuery

Now we emulate the above methodology with another query: *report course codes for courses and course codes for their prerequisites of prerequisites*. The expressions in XQuery and OOXQuery are shown below and the performance is reported in Figure 7 in the style of Figure 6.

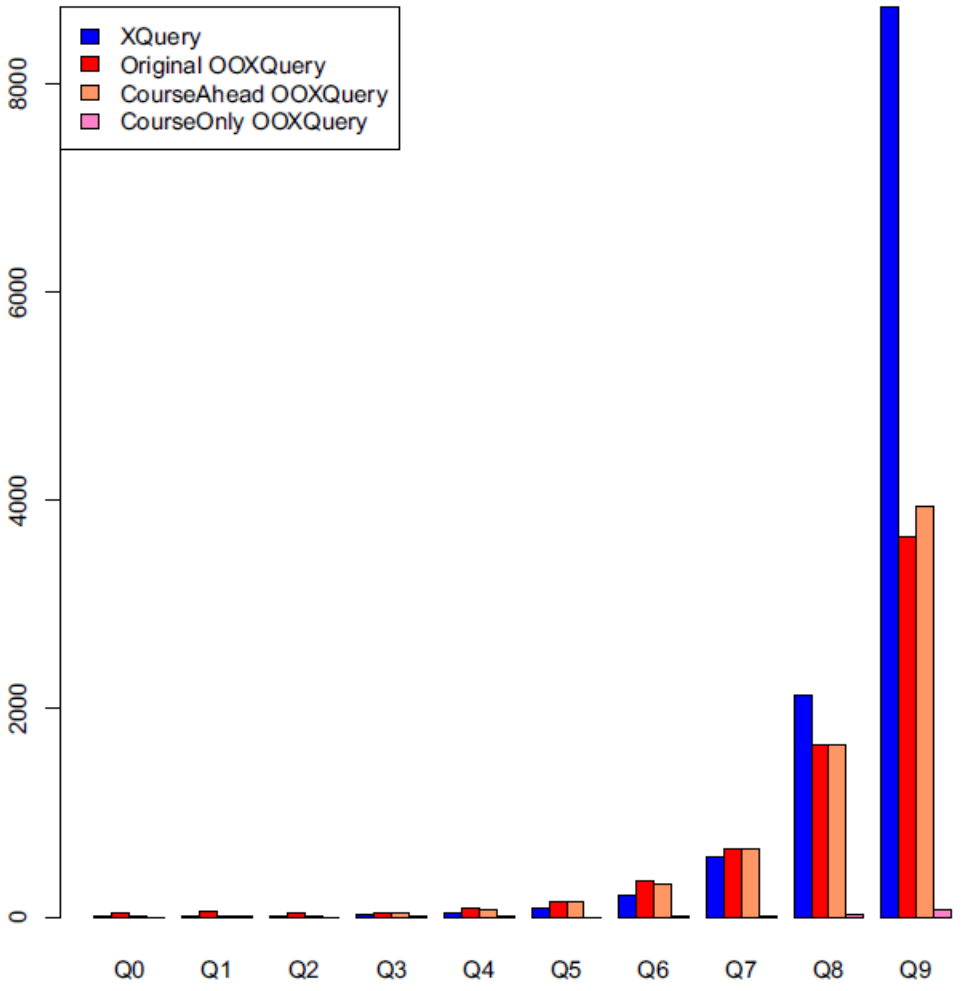

Figure 7. Prereq of Prereq

XQuery

```
for $c in doc("…/Course.xml")//Course
for $c1 in doc("…/Course.xml")//Course
where $c1/CourseCode=$c/PreReq/CourseCode
return <E> {$c/CourseCode, $c1/PreReq} </E>
```

OOXQuery

```
for $c in doc("…/UniOO-R-CourseOnly.xml")//Course
return <E> {$c.CourseCode, $c.PreReqs.PreReqs.CourseCode} </E>
```

Here too we find that OOXQuery is simpler. This time we see that the performance of XQuery is much worse than the performance of OOXQuery under scenario (iv). The situation between XQuery and OOXQuery is reversed. From this too one can conclude that the query engine could use some mechanisms to speed processing of XQuery queries. We see that the OOXQuery expressions are generally simpler. Conservatively, we can state that one should not suspect that OOXQuery will necessarily have bad performance when compared to XQuery. But, before we leave this line of thought, we consider a case where objects are a bit more intertwined and an simple fix suggested above ((iii) and (iv)) would not work and will need more attention.

3.2.2. When objects are more intertwined

In the previous section we considered Course elements that were self-contained and hence at least we could conduct an experiment by isolating them in order to hint how performance of OOXQuery could be improved by a query engine. We consider the query: *Get student's average score for each class*. The XQuery and OOXQuery expressions for it are shown below and the performance is shown in Figure 8.

XQuery

```
for $c in distinct-values(doc("…/Student.xml")//Student/Class)
let $s1:=doc("…/Student.xml")//Student[Class=$c]/GPA
return <E> {<Class>{$c}</Class>, <Ave>{avg($s1)}</Ave>} </E>
```

OOXQuery

```
for $c in distinct-values(doc("…/UniOO-R.xml")//Student/Class)
let $s:=doc("…/UniOOR1.xml")//Student[Class=$c]/GPA
return <E> {<Class>{$c}</Class>, <Ave>{avg($s)}</Ave>} </E>
```

The XQuery and OOXQuery expressions seem equally easy in this case. But even though the query only uses student information, the performance of OOXQuery is much worse than that of XQuery. As before, this can be attributed to inflation in the size of data sent to OOXQuery. However, unlike courses, the student information is more intertwined with other objects: Student objects are related to Person because of inheritance; Instructor because of mentorship; and they are also involved in Enrollment. Therefore, the student elements cannot be isolated in the OOXML document without diminishing the quality of the dataset. We hope that the implication

of the inflation of size of the dataset in OOXML does not show any inherent weakness of the proposed object-oriented approach, but lack of proper access methods and query processing mechanisms in the underling query engine.

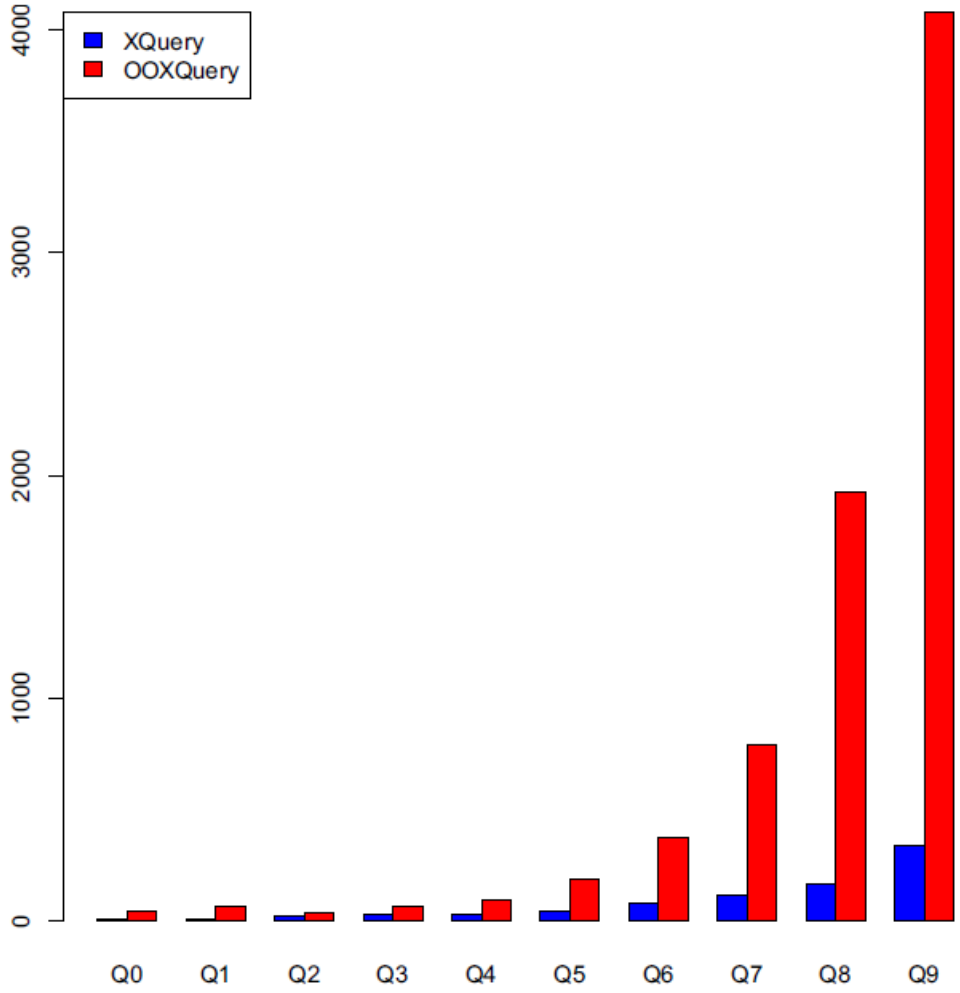

Figure 8. Students' classification and GPA

3.3. Inheritance by reference vs. inheritance by value

So far we have compared XML and OOXML. For the latter we used OOXML-R where inheritance by reference is deployed. What if we had considered OOXML-V where inheritance by value is used. We observe that we have already gained some understanding of XQuery vs. OOXQuery. Instead of repeating all benchmarking for OO-V we make direct comparison between OO-R and OO-V.

3.3.1. A simple query

We consider the query to *list the names of students*. The XQuery and OOXQuery expressions for it are shown below and the performance is shown in Figure 9.

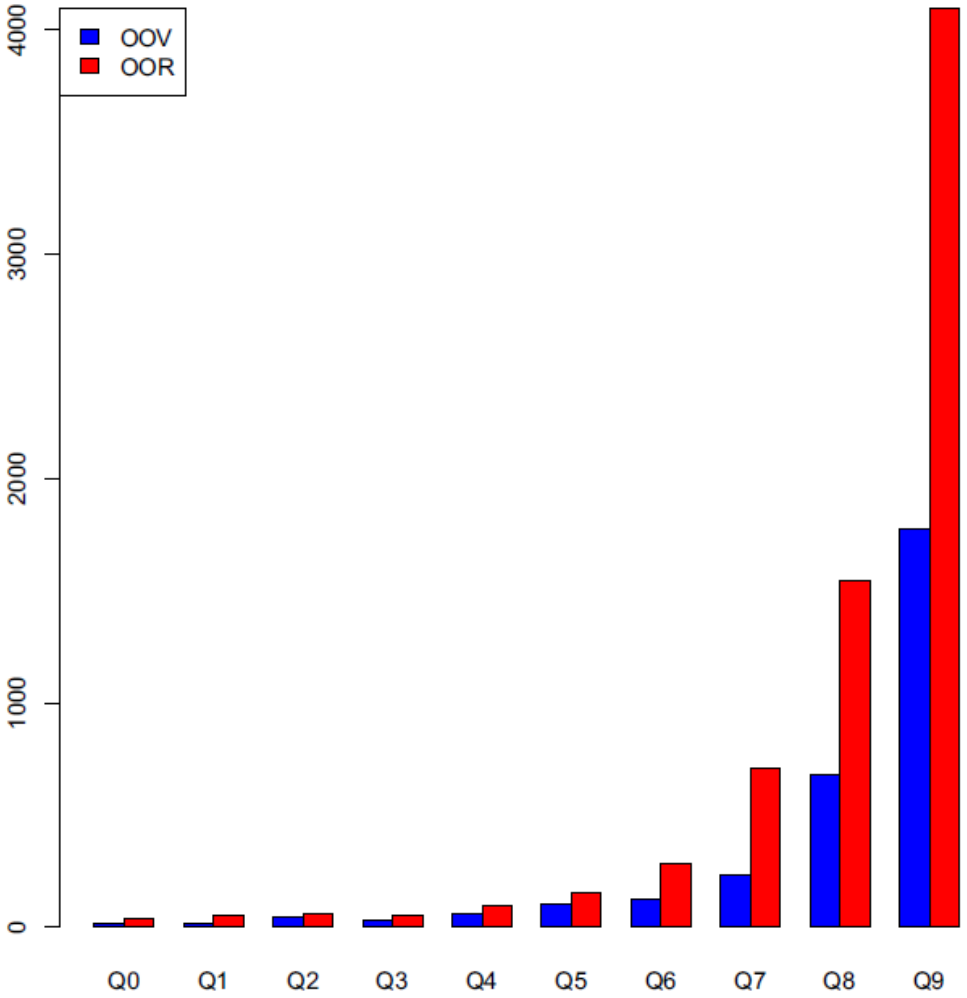

Figure 9. Names of students in OOXQuery-V vs. OOXQuery-R

OOXQuery-R with the data model OOXML-R

```
for $s in doc("…/UnivOO-R.xml")//Student
return <E> {$s.Name}
```

OOXQuery-V with the data model OOXML-V

```
for $s in doc("…/UnivOO-V.xml")//Student
return <E> {$s/Name} </E>
```

The data is organized differently in OOXML-R and OOXML-V where inheritance is involved. In case of OOXML-R the properties of super-objects are left at the super-object level and simply referenced from the object. In OOXML-V the properties from the super-objects are materialized within the object. In case of OOXML-V the super-object properties are not included in the superobject but physically materialized in the object. In our university database for example, Student as well as Instructor inherit Person. What happens if a person is a student as well as an instructor at the same time. The only choice in OOXML-V seems to be to materialize the Person properties in Student as well as Instructor objects – there by leading to duplication of

information. Also, when a student also becomes an instructor the update process would become more complex. But even from a query point of view the situation that seems simple at the level of a single object becomes more complex when we consider collections of objects, a fundamental issue we can not ignore in query languages. After all, query languages are about collection – and only about collections.

We also note a minor issue regarding syntax in OO-R vs. OO-V. Observe that in OOXML-R one uses $s**.**Name and in OOXML-V $s/Name is used. We allow the option of using $s**.**Name in the latter case as well. In this case the choice between inheritance by reference and inheritance by value does not seem to matter much. It seems that depending upon the choice between inheritance by reference vs. value, the user takes it for granted that the "**.**" or "/" understand the context clearly.

3.3.2. Query of Person in OOXQuery-R vs. OOXQuery-V

In this example we simply wish to *count the number of Person objects*. The OOXQuery-R and OOXQuery-V expressions for it are shown below and the performance is shown in Figure 10.

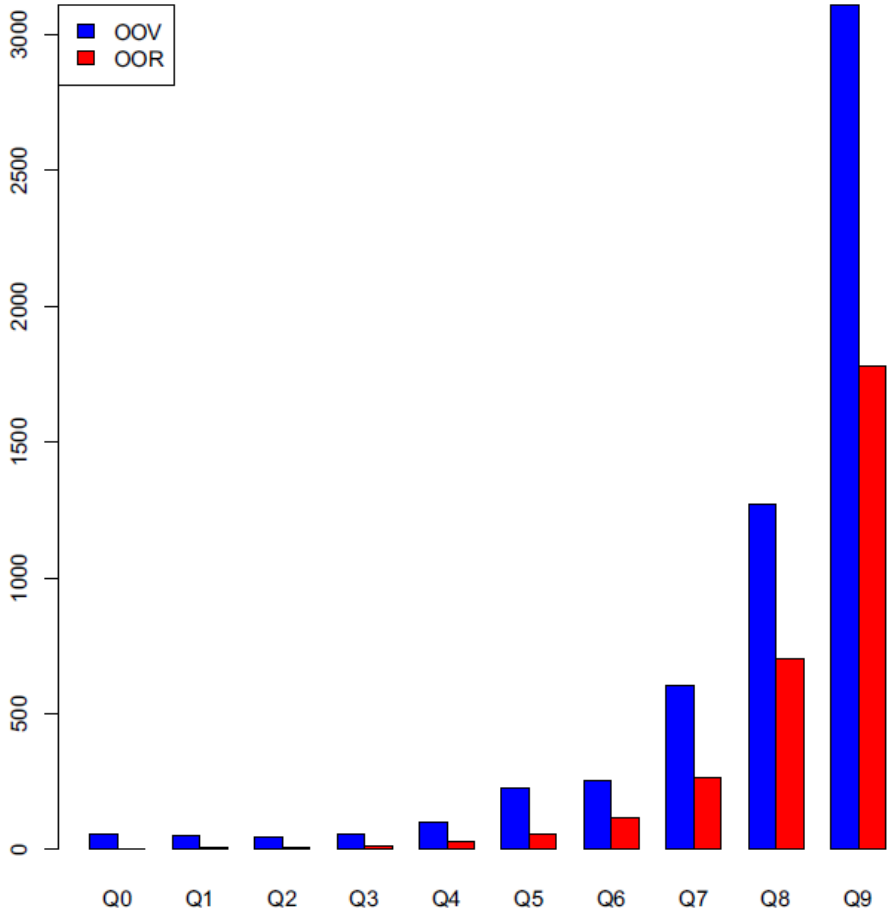

Figure 10. Counts of persons in OOXQuery-V vs. OOXQuery-R

OOXQuery-R with the data model OOXML-R

```
for $s in doc("…/UnivOO-R.xml")//University
return <E> {count($s/Person)} </E>
```

OOXQuery-V with the data model OOXML-V

```
count(for $p in doc("…/UnivOO-V.xml")//University/Person
      return $p)
+ count(for $s in doc("…/UnivOO-V.xml")//University/Student
      return $s)
+ count(for $f in doc("…/UnivOO-V.xml")//University/Faculty
      return $f)
- count(for $s in doc("…/UnivOO-V.xml")//Student
      for $f in doc("…/UnivOO-V.xml")//Faculty
       where substring($f/@FacultyID, 2)= substring($s/@StudentID, 2)
      return $f)
```

In this case XQuery-R and XQuery-V are significantly different. It is should be obvious that OOXML-R is more user friendly than OOXML-V. In this case, the performance of OOXQuery-R is also better than that of OOXQuery-V. We see that although a count is a relatively simple matter, but it becomes quite convoluted in OOXML-V. But the problems will compound with more complex queries on persons.

The preference for inheritance by reference or value may be considered a matter of individual preference. But in case of XML there is an additional issue to be considered. The concept of path expressions is important in XML. The XPath language is rather powerful foundational concept for processing XML. A path expression matches a set of nodes that can be considered a collection on its own right. Query languages in databases are all about collections. In relational databases the collections are relations. Path expressions can be considered the syntactic counterpart of relations. XQuery adds “for”, “let”, “where”, and “return” clauses to extract subsets via path expressions. Because of this inheritance by reference is more in tune with XML as it does not cause any constraints of its own on the physical placement of data.

4. Conclusion

We have benchmarked XQuery and the two versions (-R and -V) of OOXQuery. Our experiments reveal some interesting results. First, due to object-orientation and use of dotted expressions, OOXQuery seems easier to use when compared to XQuery. Likewise, OOXQuery-R seems more natural than OOXQuery-V.

The performance results are mixed. But there seem to be sufficient evidence that OOXML and OOXQuery are worthy of consideration as a serious alternative to pure XML and XQuery, respectively. The instances of differences between performance of queries in XQuery or OOXQuery engines offer opportunities to make improvements in an XQuery engine.
In our experiments we have relied on an XQuery engine that is primary meant for memory resident databases. Our experiments can be revisited as query engines become less dependent of

large amounts of main memory.

REFERENCES

[1] A proposal for an object-oriented dialect of XML. Shashi K. Gadia, Huan Lin, Yuan Yuan, and Ying Wei. March 2016.
[2] *The definition for XQuery* https://en.wikipedia.org/wiki/XQuery
[3] Peter Pin-Shan Chen. *The entity-relationship model – toward a unified view of data*. ACM Transactions on Database Systems (TODS), Vol 1, pp 9-36.
[4] Zhao, Xinyuan and Shashi K Gadia. *A Lightweight Workbench for Database Benchmarking, Experimentation, and Implementation*. IEEE Transactions on Knowledge and Data Engineering, Vol 24, 2012, pp 1937 - 1949
[5] The ODMG book.
[6] Object Data Management Group (ODMG); http://www.odbms.org/odmg-standard/
[7] XML path language XPath. http://www.w3.org/TR/xpath/

Appendix A. Extraction of XML from OOXML-R using OOXQuery

We start with BaseDoc.xml, an XML document that is designed for OOXQuery-R, and extract six ordinary XML document to be used for XQuery.

Generate Person.xml

The following does not return a legal XML document.

```
for $e in doc("…/UnivOO-R.xml")//Person
return
<Person>{
<ID>{substring($e.@PersonID,2)}</ID>,
<Name>{$e.Name/text()}</Name>,
<Address>{$e.Address/text()}</Address>,
<DOB>{$e.DOB/text()}</DOB>}
</Person>
```

Generate Faculty.xml

```
for $e in doc("…/UnivOO-R.xml")//Faculty
return
<Faculty>{
<FacultyID>{substring($e.@FacultyID,2)}</FacultyID>,
<Rank>{$e.Rank/text()}</Rank>,
<Salary>{$e.Salary/text()}</Salary>}
</Faculty>
```

Generate Student.xml

```
for $e in doc("…/UnivOO-R.xml")//Student
return
<Student>{
<StudentID>{substring($e.@StudentID,2)}</StudentID>,
<Class>{$e.Class/text()}</Class>,
<GPA>{$e.GPA/text()}</GPA>,
<MentorID>{substring($e.Mentor.@FacultyID,2)}</MentorID>,
<CreditHours>{$e.CreditHours/text()}</CreditHours>}
</Student>
```

Generate Course.xml

```
for $e in doc("…/UnivOO-R.xml")//Course
return
<Course>{
$e.CourseCode,
<CourseName>{$e.CourseName/text()}</CourseName>,
<PreReq>{$e.PreReqs.CourseCode}</PreReq>}
```

```
</Course>
```

Generate Offering.xml

```
for $e in doc("…/UnivOO-R.xml")//Offering
return
<Offering>{
<CourseCode>{$e.CourseInfo.CourseCode/text()}</CourseCode>,
<SectionNo>{$e.Section/text()}</SectionNo>,
<InstructorID>{substring($e.Instructor.@FacultyID,2)}</InstructorID>}
</Offering>
```

Generate Enrollment.xml

```
for $e in doc("…/UnivOO-R.xml")//Enrollment
return
<Enrollment>{
<CourseCode>{$e.OfferingInfo.CourseInfo.CourseCode/text()}</CourseCode>,
<SectionNo>{$e.OfferingInfo.Section/text()}</SectionNo>,
<StudentID>{substring($e.StudentInfo.@StudentID,2)}</StudentID>,
<Grade>{$e.Grade/text()}</Grade>}</Enrollment>
```

Appendix B. Extraction of OOXML-V from OOXML-R using OOXQuery

The following does not return a legal XML document.
A single OOXQuery query is needed. Faculty and Student elements inherit Person. Therefore the Person based properties are transferred from Person elements to Faculty and Student elements.
No changes are needed for

```
for $p in doc("…/UnivOO-R.xml")//Person
let $s:= doc("…/UnivOO-R.xml")//Student [ISA/@oID=$p/@PersonID]
let $f:= doc("…/UnivOO-R.xml")//Faculty [ISA/@oID=$p/@PersonID]
where count($s)= 0 and count($f)=0
return $p,
for $f in doc("…/UnivOO-R.xml")//Faculty
return
<Faculty> {
$f.@FacultyID,
$f.Name,
$f.Address,
$f.DOB,
$f.Salary,
$f.Rank,
$f/Mentoring,
$f/Instructing}
</Faculty>,
for $s in doc("…/UnivOO-R.xml")//Student
return
<Student> {
$s.@StudentID,
$s.Name,
$s.Address,
$s.DOB,
$s.Class,
$s.CreditHours,
$s.GPA,
$s/Mentor,
$s/EnrolledIn}
</Student>,
for $c in doc("…/UnivOO-R.xml")//Course
return $c,
```

```
for $o in doc("…/UniOO-R.xml")//Offering
return $o,
for $e in doc("…/UnivOO-R.xml")//Enrollment
return $e
```

Appendix C. Using OOXQuery for generating commands for creation of nodes and edges in a Neo4j database using OOXQuery

Neo4j supports a data model and a query language (Cypher) that are graph-based, consisting of nodes and edges. The data model in Neo4j has a great deal of similarity with OOXML. The nodes
in Neo4j can be considered counterpart of entities in OOXML and relationships in Neo4j can be considered counterpart of relationships in OOXML. However, the query languages Cypher and OOXQuery differ significantly. In OOXML and OOXQuery the schema has to be known ahead of
time. A given dataset in OOXML can be converted to a dataset in Neo4j quite easily using OOXQuery. First and OOXQuery query can be used to return create statements. These statements
are then executed in Cypher to create nodes. Similarly, another OOXQuery is executed to output match statements for creation of edges. The following example shows how the OOXML-based UniversityDB database is loaded in Neo4j.

Creation of nodes

```
for $p in doc("…/UniOO-R.xml")//Person
return
<E> {
concat("create (",
substring($p.@PersonID, 1),
":Person { ID:",
substring($p.@PersonID, 2),", Name: '",
$p.Name, "', Address: '",
$p.Address, "', DOB: '",
$p.DOB,"' })")}
</E>
,
for $f in doc("…/UniOO-R.xml")//Faculty
return
<E> {
concat("create (",
substring($f.@FacultyID, 1),
":Faculty { ID:", substring($f.@FacultyID, 2),",
Salary: ", $f.Salary, ",
Rank: '", $f.Rank, "' })")} </E>
,
for $s in doc("…/UniOO-R.xml")//Student
return
<E> {
concat("create (",
substring($s.@StudentID, 1),
":Student { ID:", substring($s.@StudentID, 2),",
Class: '", $s.Class, "',
CreditHours: ", $s.CreditHours, ",
GPA: ", $s.GPA," })")}
</E>
,
```

```
for $c in doc("…/UniOO-R.xml")//Course
return
<E> {
concat("create (",
substring($c.@CourseID, 1), ":Course { ID:",
substring($c.@CourseID, 3),", CourseCode: '", $c.CourseCode, "',
CourseName: '", $c.CourseName,"'})")}
</E>
,
for $d in doc("…/UniOO-R.xml")//Offering
return
<E> {
concat("create (",
substring($d.@OfferingID, 1), ":Offering { ID:", substring($d.@OfferingID, 10),",
Section: ", $d.Section, "})")}
</E>
```

Creation of relationships

```
for $s in doc("…/UniOO-R.xml")//Student
return
<E> {
concat("match ( a:Student), (b:Person)
where a.ID = ",substring($s.@StudentID, 2), "
and b.ID =", substring($s.@StudentID, 2),"
create (a) - [r:s2p]-> (b)")}
</E>
,
for $f in doc("…/UniOO-R.xml")//Faculty
return
<E> {
concat("match ( a:Faculty), (b:Person)
where a.ID = ",substring($f.@FacultyID, 2), "
and b.ID =", substring($f.@FacultyID, 2),"
create (a) - [r:f2p]-> (b)")}
</E>
,
for $d in doc("…/UniOO-R.xml")//Student
return
<E> {concat("match ( a:Faculty), (b:Student)
where a.ID = ",substring($d.Mentor.@FacultyID, 2), "
and b.ID =", substring($d.@StudentID, 2),"
create (a) - [r:mentor]-> (b)")} </E>
,
for $c in doc("…/UniOO-R.xml")//Offering
return
<E>
{concat("match ( a:Faculty), (b:Offering)
where a.ID = ",substring($c.Instructor.@FacultyID, 2), "
and b.ID =", substring($c.@OfferingID, 10),"
create (a) - [r:teach]-> (b)")}
</E>
,
for $o in doc("…/UniOO-R.xml")//Offering
return
<E>
{concat("match ( a:Offering), (b:Course)
where a.ID = ",substring($o.@OfferingID, 10), "
and b.ID =", substring($o.CourseInfo.@CourseID, 3),"
create (a) - [r:belong]-> (b)")}
</E>
,
```

```
for $s1 in doc("…/UniOO-R.xml")//Course
return
<E> {
concat("match ( a:Course), (b:Course)
where a.ID = ",substring($s1.@CourseID, 3), "
and b.ID =", substring (string-join( ($s1.PreReqs.@CourseID), ''),3,3),"
create (a) - [r:preReqs]-> (b)")} </E>
,
for $s2 in doc("…/UniOO-R.xml")//Course
return <E> {concat("match ( a:Course), (b:Course) where a.ID =
",substring($s2.@CourseID, 3), " and b.ID =", substring (string-join(
($s2.PreReqs.@CourseID), ''),8,3)," create (a) - [r:preReqs]-> (b)")} </E>
,
for $e in doc("…/UniOO-R.xml")//Enrollment
return
<E> {
concat("match ( a:Student), (b:Offering)
where a.ID = ",substring($e.StudentInfo.@StudentID, 2), "
and b.ID =", substring($e.OfferingInfo.@OfferingID, 10), "
create (a) - [r:take{ID:'",$e.@EnrollmentID, "',Grade:'",$e.Grade,"'}]-> (b)")}
</E>
```

Appendix D. Using OOXQuery for generating dataset for an SQL database

```
//Get person;
<showtext>
$OOXQuery:>
<Relation name="Person">
<Columns>
<Column name="Name" type="Text"/>
<Column name="ID" type="Text"/>
<Column name="Address" type="Text"/>
<Column name="DOB" type="Date"/>
</Columns>
{
for $e in doc("…/UniOO-R.xml")//Person
return <Tuple>{<Name>{$e.Name/text()}</Name>,<ID>{substring($e.@PersonID,2)}</ID>,
<Address>{$e.Address/text()}</Address>,<DOB>{$e.DOB/text()}</DOB>}</Tuple>
} </Relation>;
</showtext>
//Get Instructor;
<showtext>
$OOXQuery:>
<Relation name="Instructor">
<Columns>
<Column name="InstructorID" type="Text"/>
<Column name="Rank" type="Text"/>
<Column name="Salary" type="Integer"/>
</Columns> {
for $e in doc("…/UniOO-R.xml")//Faculty
return <Tuple>{<InstructorID>{substring($e.@FacultyID,2)}</InstructorID>,
<Rank>{$e.Rank/text()}</Rank>,<Salary>{$e.Salary/text()}</Salary>}</Tuple>
} </Relation>;
</showtext>
//Get Student
;
<showtext>
$OOXQuery:>
<Relation name="Student">
<Columns>
```

```
<Column name="StudentID" type="Text"/>
<Column name="Class" type="Text"/>
<Column name="GPA" type="Double"/>
<Column name="MentorID" type="Text"/>
<Column name="CreditHours" type="Integer"/>
</Columns> {
for $e in doc("…/UniOO-R.xml")//Student
return <Tuple>{<StudentID>{substring($e.@StudentID,2)}</StudentID>, <Class>{$e.Class/
text()}</Class>,<GPA>{$e.GPA/text()}</GPA>,
<MentorID>{substring($e.Mentor.@FacultyID,2)}</MentorID>,<CreditHours>{$e.CreditHours/
text()}</CreditHours>}</Tuple>
} </Relation>;
</showtext>
//Get Course
;
<showtext>
$OOXQuery:>
<Relation name="Course">
<Columns>
<Column name="CourseCode" type="Text"/>
<Column name="CourseName" type="Text"/>
<Column name="PreReq" type="Text"/>
</Columns> {
for $e in doc("…/UniOO-R.xml")//Course
return <Tuple>{<CourseCode>{$e.CourseCode/text()}</CourseCode>,
<CourseName>{$e.CourseName/text()}</CourseName>,<PreReq>{$e.PreReqs.CourseCode/
text()}</PreReq>}</Tuple>
} </Relation>;
</showtext>
//Get Offering
;
<showtext>
$OOXQuery:>
<Relation name="Offering">
<Columns>
<Column name="CourseCode" type="Text"/>
<Column name="SectionNo" type="Integer"/>
<Column name="InstructorID" type="Text"/>
</Columns> {
for $e in doc("…/UniOO-R.xml")//Offering
return <Tuple>{<CourseCode>{$e.CourseInfo.CourseCode/text()}</CourseCode>,
<SectionNo>{$e.Section/text()}</
SectionNo>,<InstructorID>{substring($e.Instructor.@FacultyID,2)}</InstructorID>}</
Tuple>
} </Relation>;
</showtext>
//Get Enrollment
;
<showtext>
$OOXQuery:>
<Relation name="Enrollment">
<Columns>
<Column name="CourseCode" type="Text"/>
<Column name="SectionNo" type="Integer"/>
<Column name="StudentID" type="Text"/>
<Column name="Grade" type="Text"/>
</Columns>
{
for $e in doc("…/UniOO-R.xml")//Enrollment
return <Tuple>{<CourseCode>{$e.OfferingInfo.CourseInfo.CourseCode/text()}</
CourseCode>, <SectionNo>{$e.OfferingInfo.Section/text()}</
SectionNo>,<StudentID>{substring($e.StudentInfo.@StudentID,2)}</
```

```
StudentID>,<Grade>{$e.Grade/text()}</Grade>}</Tuple>
} </Relation>;
</showtext>
```

Appendix E： Simplified version of the benchmark

```
// Clear the variable environment to re-execute in the same CyDIW session;
$CyDB:> undeclare string $$osType;
$CyDB:> declare string $$osType;
$CyDB:> set $$osType := Windows;
$OS:> del CyWorkspace\benchmarkSubset.xml;
$CyDB:> undeclare string $$Project_R_Is_Installed;
$CyDB:> undeclare string[] $$prefix;
$CyDB:> undeclare string[] $$query;
$CyDB:> undeclare string $$xmldoc;
$CyDB:> undeclare int $$i;
$CyDB:> undeclare int $$j;
$CyDB:> undeclare int $$k;
$CyDB:> if ($$osType == "Windows") {$OS:> del CyWorkspace\Plot.pdf;}
else {$OS:> rm CyWorkspace\Plot.pdf;}
// Verify that the variable environment is clear;
$CyDB:> list variables;
// Verify if Project R is available for reporting performance graphs;
$CyDB:> declare string $$Project_R_Is_Installed;
$CyDB:> set $$Project_R_Is_Installed := Yes;
// If R is not installed, No can be used, and the much of the experiment can be run;
// Step 1. Create variables;
// String variables to hold query engine such as OOXQuery, XQuery;
$CyDB:> declare string[2] $$prefix;
// String variables to hold queries;
$CyDB:> declare string[11] $$query;
$CyDB:> declare string $$xmldoc;
$CyDB:> declare int $$i;
$CyDB:> declare int $$j;
$CyDB:> declare int $$k;
$CyDB:> declare int $$n;
$CyDB:> set $$n := 7;
// Create benchmarkTime.xml, an XML-based log file, to gather performance statistics;
$CyDB:> createLog <root> benchmarkTime.xml;
// The file uses user defined <root> as its root;
// Assign queries to array variables;
$CyDB:> set $$prefix[0] := $XQuery;
$CyDB:> set $$query[0] :=
for $p in doc("C:/Ying_Wei/newDataSet/UniXML/0/Person.xml")//Person
for $i in doc("C:/Ying_Wei/newDataSet/UniXML/0/Faculty.xml")//Faculty
where $i/Salary>=100000 and $i/FacultyID=$p/ID
return <E> { $p/Name, $i/Rank} </E>;
$CyDB:> list variables;
/*
Step 2. Query execution and logging of benchmark statistics.
Create Populate the xml log file with performance stats.
Outer loop for different query engines is tagged <loop1>
For every query, the query is run 3 times to warm-up the caches
In an inner loop tagged <query>, query is run performance stat is reported
*/
$CyDB:> foreach $$i in (0) log time >> <loop1 var="XQuery"> benchmarkTime.xml
{
// Inner loop to iterate through XMark queries;
$CyDB:> foreach $$j in [0, $$n] log time >> <loop2 var="Q($$j)"> benchmarkTime.xml
{
// Warm up caches with 3 executions - don't record the performance yet;
$CyDB:> foreach $$k in [1,3]
```

```
{$CyDB:> run $$prefix[$$i] $$query[$$j]; }
// Now execute each query and log the performance;
$CyDB:> foreach $$k in [1,3]
{$CyDB:> run $$prefix[$$i] $$query[$$j]
out>> ($$prefix[$$i])_query($$j).xml
log time >> <query> benchmarkTime.xml;
}
}
}
$CyDB:> undeclare string[] $$OOquery;
$CyDB:> list variables;
// Step 1. Create variables;
$CyDB:> declare string[11] $$OOquery;
$CyDB:> undeclare string $$OOXQuery0;
$CyDB:> declare string $$OOXQuery0;
$CyDB:> undeclare string $$Query0Translation;
$CyDB:> declare string $$Query0Translation;
$CyDB:> set $$OOXQuery0:=
<Item xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> {
for $f in doc("C:/Ying_Wei/newDataSet/UniOO_R/UniOOR1.xml")//Faculty
where $f/Salary>=100000
return <E> { $f.Name, $f.Rank} </E>
} </Item>;
$OOXQuery:> translate $$OOXQuery0 $$Query0Translation;
$CyDB:> set $$OOquery[0] :=
$$Query0Translation;
$CyDB:> list variables;
$XQuery:> setStrictSchemaValidation true;
// Step 2. Query execution and logging of benchmark statistics;
$CyDB:> foreach $$i in (0) log time >> <loop1 var="OOXQuery"> benchmarkTime.xml
{
// Inner loop to iterate through XMark queries;
$CyDB:> foreach $$j in [0, $$n]
log time >> <loop2 var="Q($$j)"> benchmarkTime.xml
{
// Warm up 3 times - no stats collected;
$CyDB:> foreach $$k in [1,3]
{$CyDB:> run $$prefix[$$i] $$OOquery[$$j];}
// Now execute each query and log performance;
$CyDB:> foreach $$k in [1,3]
{$CyDB:> run $$prefix[$$i] $$OOquery[$$j]
out>> ($$prefix[$$i])_OOXQuery($$j).xml
log time >> <query> benchmarkTime.xml;
}
}
}
$XQuery:> setStrictSchemaValidation false;
// Step 3. Display full contents of log file;
$CyDB:> displayFile CyWorkspace/benchmarkTime.xml;
// Step 4. Calculate and display benchmarkSubset.xml for recording the average
execution time for each engine and query;
$XQuery:> for $e in doc("CyWorkspace/benchmarkTime.xml")//loop1
return <loop1> {
$e/@var,
for $f in $e/loop2
return <loop2> {
$f/@var,
let $g := $f/query/text()
return <avg> {avg($g)} </avg>
} </loop2>
} </loop1>
out >> benchmarkAvg.xml;
$CyDB:> displayFile CyWorkspace/benchmarkAvg.xml;
```

```
// Step 5. If Project R has been installed, compute and display the performance graph
othewise display a sample graph;
$CyDB:> if ($$Project_R_Is_Installed == "Yes")
{// Display the specifications for the graph to be passed to R
$CyDB:> displayFile cyclients/r/workspace/R_code.txt;
// Now use R to create the graph in pdf format
$R:> CMD BATCH cyclients/r/workspace/R_codeForXQueryAndOOXQuery.txt;
// Display the computed plot (Plot.pdf is mentioned in the R_code;
$CyDB:> displayPDF CyWorkspace/PlotTime.pdf;
} else {// Display sample plot;
$CyDB:> displayPDF ComS363\Demos\Datasets\SamplePlot.pdf;
}
```