Universität Freiburg
Institut für Informatik
Prof. Dr. Peter Fischer
Lecture XML and Databases
Winter Semester 2011/12

Due date/discussion: 3.2.2012

Solution Sheet 10

Query Rewrites, Node IDs

Exercise 1:

1.1. No, although the results look the same. In the rewrite, the two nodes do not have the same identity, whereas in the original version they do.

Look at the following counterexample:

```
let $a:=<a><b/></a>
let $c:= ($a, $a)
return $c/b
```

returns <b/> because the step/ performs duplicate elimination, whereas

```
let $c := (<a><b/></a>, <a><b/></a>)
return $c/b
```

returns <b/>, <b/>.

1.2.
If one executed the queries in a naive way without additionally sorting and eliminating duplicates, then an invocation of //a/b on

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <a>
        <a><b id="1"/></a>
        <b id="2"/>
    </a>
</root>
```

would return <b id="2"/> (children of the first <a/>) and then <b id="1"/> (child of the second <a/>). Hence, sorting has to be explicitely done to respect the XQuery specification.

1.3.
In the first case, since we are comparing two sequences of one single item, each of them being a string, it is possible to replace **=** with **eq**.

In the second case, additional information is necessary, i.e., in a schema. For the replacement to be possible, one needs to know that there is exactly one author. This is also a sufficient condition: if the author is a string, but also if it is a numeric value (in the latter case, an error is raised for = as well as for eq).

Exercise 2: Node IDs

2.1. Note that every node gets a node ID, including text nodes.
a. (Integer IDs). In this example, we start at 1.

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [2]<Passenger>
        [3]<name>[4]Santa Claus</name>
        [5]<passnumber>[6]000112</passnumber>
        [7]<address>[8]Somewhere</address>
    </Passenger>
    [9]<Reservation>
        [10]<date>[11]2006-12-24</date>
        [12]<flightRef>[13]LX124</flightRef>
        [14]<passRef>[15]000111</passRef>
    </Reservation>
</doc>
```

b. (Double IDs). This is the same as integers, but with 1.0, 2.0...

```
<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
    [2.0]<Passenger>
        [3.0]<name>[4.0]Santa Claus</name>
        [5.0]<passnumber>[6.0]000112</passnumber>
        [7.0]<address>[8.0]Somewhere</address>
    </Passenger>
    [9.0]<Reservation>
        [10.0]<date>[11.0]2006-12-24</date>
        [12.0]<flightRef>[13.0]LX124</flightRef>
        [14.0]<passRef>[15.0]000111</passRef>
    </Reservation>
</doc>
```

c. (Dewey IDs). Root is one, and for children we add a level.

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.2]<passnumber>[1.1.2.1]000112</passnumber>
        [1.1.3]<address>[1.1.3.1]Somewhere</address>
    </Passenger>
    [1.2]<Reservation>
        [1.2.1]<date>[1.2.1.1]2006-12-24</date>
        [1.2.2]<flightRef>[1.2.2.1]LX124</flightRef>
        [1.2.3]<passRef>[1.2.3.1]000111</passRef>
    </Reservation>
</doc>
```

d. (ORDPATH IDs). This is the same as Dewey, but even numbers do not count as a level.

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
        [1.1.5]<address>[1.1.5.1]Somewhere</address>
    </Passenger>
    [1.3]<Reservation>
        [1.3.1]<date>[1.3.1.1]2006-12-24</date>
        [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
        [1.3.5]<passRef>[1.3.5.1]000111</passRef>
    </Reservation>
</doc>
```

2.2. b. We begin with double IDs. We distribute the IDs evenly between 8 and 9:

```
<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
    [2.0]<Passenger>
        [3.0]<name>[4.0]Santa Claus</name>
        [5.0]<passnumber>[6.0]000112</passnumber>
        [7.0]<address>[8.0]Somewhere</address>
    </Passenger>
    [8.125]<Reservation>
        [8.25]<date>[8.375]2008-12-26</date>
        [8.5]<flightRef>[8.625]LX183</flightRef>
        [8.75]<passRef>[8.875]000111</passRef>
    </Reservation>
      [9.0]<Reservation>
        [10.0]<date>[11.0]2006-12-24</date>
        [12.0]<flightRef>[13.0]LX124</flightRef>
        [14.0]<passRef>[15.0]000111</passRef>
    </Reservation>
</doc>
```

d. For ORDPATH, the trick is to use even numbers. We assign to the inserted node the ID 1.2.1 where 2 does not count as a level (so 1.2.1 is at the second level, its siblings are 1.1 and 1.3) We have 1.1 < 1.1.5.1 < 1.2.1 < 1.3, so that the document order is maintained. It is useless to renumber the IDs after the updates.

```
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
    [1.1]<Passenger>
        [1.1.1]<name>[1.1.1.1]Santa Claus</name>
        [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
        [1.1.5]<address>[1.1.5.1]Somewhere</address>
    </Passenger>
    [1.2.1]<Reservation>
        [1.2.1.1]<date>[1.2.1.1.1]2008-12-26</date>
        [1.2.1.3]<flightRef>[1.2.1.3.1]LX183</flightRef>
        [1.2.1.5]<passRef>[1.2.1.5.1]000111</passRef>
    [1.3]<Reservation>
        [1.3.1]<date>[1.3.1.1]2006-12-24</date>
        [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
        [1.3.5]<passRef>[1.3.5.1]000111</passRef>
    </Reservation>
</doc>
```

a. For Integer IDs and Dewey IDs, there is no room to add new IDs. There are three possibilities to overcome this:

- we renumber all IDs whenever an update occurs

- we leave gaps in the initial renumbering (e.g. 1001, 2001...). When inserting a new node, we split the space evenly between two adjacent IDs : for example, to add 7 IDs between 1001 and 2001: 1125, 1250, 1375, 1500, 1625, 1750, 1875. When the gaps become too small, a renumbering is necessary.
- we add an additional structure which maintains the connections (parent, child, sibling). However, this decreases the advantage that integer IDs provide the best way for document traversal.

1.3. This is a rough estimate on the properties of the IDs. It is the application pattern which decides which type of IDs are used (many updates Vs. many queries). Additional indexes can compensate for weaker IDs, so that there is no best solution.

	Size	Doc order (next)	Ancestor/Descendant	Sibling/Before/After node (navigation)	Insert	Computation intensity
Integer IDs	+	+				+
double IDs	+	+			+	+
Dewey IDs		+	+	+		
ORDPATH IDs	(+)	+	+	+	+	(+)