201300180 Data & Information – Test 3 ANSWERS

3 June 2015, 13:45 – 15:15

WARNING: The answers below are not complete, i.e., there may be many more correct and almost correct answers. Furthermore, the assessment is not based on a number of points per sub-question, but on the level of mastery on a certain number of aspects.

Question 1 (XQuery) (35 points)

a) Give all titles of entries that were validated.

```
/entry[@validated="1"]/title
//title[../@validated="1"]
```

b) Give all entries that are about the term “TCP/IP”.

```
//entry[.//term='TCP/IP']
//term[.=’TCP/IP’]/ancestor::entry
```

c) Give the count of how many entries have more than one review.

```
count(/entry[count(./review)>1])
count(/entry[./review[2]])
```

d) For each unique term, give the number of reviews with that term.

```
for $t in distinct-values(//term)
let $revs := //entry[.//term = $t]
return <term name="{$t}">{count($revs)}</term>

for $rev in /entry
for $t in $rev//term
group by $t
return <term name="{$t}">{count($rev)}</term>
```

The question is a bit ambiguous in what to count: entry elements or review elements. This is also correct:

```
for $rev in //review
for $t in $rev/term
group by $t
return <term name="{$t}">{count($rev)}</term>
```

e) Give the entry that comes after the one with title “Data on the Web”

```
/entry[title=”Data on the Web”]/following-sibling::entry[1]
/entry[preceding-sibling::entry[1]/title=”Data on the Web”]
```

f) Produce an XML document with root element “index”, its children are all terms, i.e., “term” elements with each an attribute “name” containing the term. The term elements have as children the titles of all entries with that term.

```
<index>{
for $t in distinct-values(//term)
let $e := /entry[.//term=$t]
return <term name="{$t}">{$e/title}</term>
}</index>
```

g) Insert a review with some made-up text to the entry about “TCP/IP Illustrated”

```
let $e := //entry[title='TCP/IP Illustrated']
let $rev := <review>foo bar</review>
return insert node $rev into $e
```

h) Set attribute “validated” of entry “Advanced Programming in the Unix environment” to “1”.

```
let $e := //entry[title='Advanced Programming in the Unix environment']
return replace value of node $e/@validated with '1'

let $e := //entry[title='Advanced Programming in the Unix environment']
return replace node $e/@validated with attribute {‘validated’} {‘1’}
```

i) Can an XML document be valid but not well-formed? Explain your answer.

Well-formed means that it is syntactically correct including, for example, that the opening and closing tags match and are properly nested. Valid means that it conforms to the given schema. In other words, for a document to be valid, it needs to be well-formed first! Otherwise it is not proper XML and cannot be validated against the schema.
So, NO, it cannot.

j) Is any XQuery function that produces (X)HTML, a RESTXQ function? Explain your answer.

k) No. It should also have annotations like %rest:path.

l) *Another direction one could have answered is*: No. RESTXQ is meant for computer-readability and data exchange by applications; (X)HTML is for human readability.

m)

Question 2 (Database transactions) (35 points)

a) The table definitions below contain primary and foreign key declarations. Suppose the database contains several entries (including one with eid 6234) and for each entry there exist one or more reviews. What happens if you issue the command “DELETE FROM entry WHERE eid=6234”? Explain your answer.

```
CREATE TABLE entry (
  eid INT,
  title TEXT,
  validated CHAR,
  PRIMARY KEY (id)
)
```

```
CREATE TABLE review (
        rid INT,
        txt TEXT,
        eid INT,
        PRIMARY KEY (rid),
        FOREIGN KEY (eid) REFERENCES entry(eid)
      )
```

It will refuse to delete the entry. The foreign key declaration creates a constraint that makes sure that if there is a review, then the entry it refers to with its eid-attribute, will also always exist. Deleting an entry for which a review exists would violate this, so it is refused.

b) Give the CREATE VIEW statement(s) for providing a certain set of users only access to validated entries including only the reviews of validated entries.

```
CREATE VIEW validated_entry AS
   SELECT * FROM entry WHERE validated="1"
CREATE VIEW validated_review AS
   SELECT review.* FROM review,validated_entry WHERE review.eid=validated_entry.eid
```

c) Given the SQL statement below. How many write locks are obtained for this statement?

```
UPDATE entry
SET validated = '1'
WHERE eid IN (SELECT eid FROM review)
```

The WHERE-clause is true for any entry that has at least one review. Each of these entries will be updated, so as many write locks are obtained as there are entries with at least one review.

d) Given the schedule below. Which pairs of operations in the schedule are conflicting?

$w_1(x)\ r_2(y)\ w_1(y)\ r_1(z)\ r_3(x)\ r_3(y)\ r_3(z)$

$w_1(x)$ conflicts with $r_3(x)$
$r_2(y)$ conflicts with $w_1(y)$
$w_1(y)$ conflicts with $r_3(y)$

NOT: $r_1(z)$ with $r_3(z)$ (because both are reads)

e) Is the schedule above serializable or not? Explain why.

The first conflict dictates that T1 should be before T3
The second conflict dictates that T2 should be before T1
The third conflict dictates that T1 should be before T3

So, there is a serial schedule that keeps the conflicting operations in the same order as they are now, namely, T2 T1 T3 (i.e., $r_2(y)\ w_1(x)\ w_1(y)\ r_1(z)\ r_3(x)\ r_3(y)\ r_3(z)$). Therefore, yes, it is serializable.

f) Suppose your Java-program connects to the database and performs the following transaction: it reads one particular entry and counts the number of reviews about it, prepares some HTML-template for the entry, and then reads the reviews from the database and adds them to the template. What is the lowest isolation level for this transaction to run guaranteed correctly? Explain your answer?

The transaction reads the reviews twice, first for counting, then for adding them to the template. It should not happen that in between these two reads, another transaction is able to add or remove a review (which is called a *phantom*). Therefore, we need SERIAZABLE, because this is the isolation level that protects against phantoms.

Question 3 (REST) (40 points)

a) Annotation `@Path("/somePath")` is used to define which class should be selected for a certain URI (`/somePath` in this case). Annotations `@GET`, `@POST`, `@PUT` and `@DELETE` denote the HTTP request methods, and are combined with the expected and generated encoding defined with `@Produces` and `@Consumes` for the selection of the method of the class that is executed to handle the HTTP request message.
`@Path("/someOtherPath")` can also be used to define that a method is executed for some more specific path (`/somePath/someOtherPath` in this example for a method of the class that already answers to `/somePath`).

Example (from lecture PROG 3)

```
@Path("/hello")
public class Hello {
   @GET
   @Produces(MediaType.TEXT_PLAIN)
   public String sayPlainTextHello() {
       return "Hello Jersey";
   }
   @GET
   @Produces(MediaType.TEXT_XML)
   public String sayXMLHello() {
       return "<xml version=\"1.0\"?>\n<hello>HelloJersey</hello>";
   }
}
```

In this example, annotation `@Path("/hello")` indicates that HTTP request messages sent to `/hello` are forwarded to class `Hello`, and annotations `@GET` and `@Produces(MediaType.TEXT_XML)` indicate that GET HTTP request messages that accept XML encoding are processed by method `sayXMLHello()`.

b) POST HTTP message to address `http://anyhost/profiles` or `http://anyhost/profiles/id`, depending if the id is determined by the service or the client, containing a representation of the profile in the encoding indicated in the message encoding header. The service returns a HTTP response message indicating that the creation was successful possibly containing the representation of the profile encoded in the encoding indicated in the accept message header of the HTTP request.

c) A RESTful service in Jersey/JAX-RS is deployed in a container that offers a runtime environment for the service. The Jersey/JAX-RS container is itself a servlet that runs inside an application server. Tomcat is an application server and is used to run the Jersey container, allowing web services deployed in this container to interact with their clients.