# Data & Information – Solutions to Test 3

## Question 1

a) SELECT entry.eid, entry.title
FROM entry, review
WHERE entry.eid = review.eid
AND to_tsvector('english',review.txt) @@ to_tsquery('english', 'awful | horrible')

b) It is an "aggregate" function. It takes all values of 'txt' for a group of records (in this case, the group is formed by all records with the same eid), and concatenates them.

c) For documents 1, 6, 8, 11, 12, 14: tf(d,t) = 0 for all terms
tf(3,my)=tf(4,my)=tf(5,my)=tf(7,my)=tf(9,my)=tf(10,my)=1; tf(13,my)=2; otherwise tf(d,my)=0
tf(2,limbs)=tf(13,limbs)=1; otherwise tf(d,limbs)=0
tf(4,work)=2; otherwise tf(d,work)=0
df(my) = 7, so idf(my) = log(14/7) = log(2) = ±0.3
df(limbs) = 2, so idf(limbs) = log(14/2) = log(7) = ±0.845
df(work) = 1, so idf(work) = log(14/1) = log(14) = ±1.146
Hence, Rank(d,q) = 0 for d=1,6,8,11,12, and 14;
Rank(2,q) = 1*0.845 = 0.845
Rank(d,q) = 1*0.3 = 0.3 for d=3,5,7,9, and 10
Rank(4,q) = 1*0.3 + 2*1.146 = ±2.593
Rank(13,q) = 2*0.3 + 1*0.845 = ±1.447
Hence the most relevant document is 4, then 13, then 2, etc.

An alternative solution may also be correct: If you take not the lines as individual documents, but the whole poem, then the calculation becomes as below.

tf(1,my)=8; tf(1,limbs)=2; tf(1,work)=2
df(my) = df(limbs) = df(work) = 1 => idf(t) = log(1/1) = 0 for all terms
Rank(d,q) = 8*0 + 2*0 + 2*0 = 0

d) P(my,limbs | 13) = P(my | 13) * P(limbs | 13) = 2/10 * 1/10 = 2/100 = 0.02.

## Question 2

a) 1: $r_1(x)$ and $w_2(x)$
2: $r_1(y)$ and $w_3(y)$
3: $w_2(x)$ and $w_1(x)$
4: $w_3(y)$ and $r_4(y)$

b) T1 should be before T2 because of conflict 1
T2 should be before T3 because of conflict 2
T2 should be before T1 because of conflict 3
T3 should be before T4 because of conflict 4
The first and third cannot be true at the same time, hence not serializable

c) Dirty write: $w_1(x)$, because it writes to 'x' while T2 had previously written to 'x'
Dirty read: $r_4(y)$, because it reads 'y' while T3 had previously written to 'y'

d) 1: $r_1(x)$: request by T1 for read lock on 'x' granted; T1 reads 'x'
2: $r_1(y)$: request by T1 for read lock on 'y' granted; T1 reads 'y'
3: $w_2(x)$: request by T2 for write lock on 'x' not granted (T1 holds read lock on 'x'); T2 is delayed.
4: $w_3(y)$: request by T3 for write lock on 'y' not granted (T1 holds read lock on 'y'); T3 is delayed.
5: $w_1(x)$: request by T1 for write lock on 'x' granted; T1 writes to 'x'
6: $r_4(y)$: request by T4 for read lock on 'y' granted (nobody holds a write lock on 'y'); T4 reads 'y'
7: $c_1$: T1 commits and releases its locks; T2 and T3 re-activated
8: $w_2(x)$: request by T2 for write lock on 'x' granted; T2 writes to 'x'
9: $w_3(y)$: request by T3 for write lock on 'y' again not granted (T4 holds read lock on 'y'); T3 is delayed again.
10: $c_2$ : T2 commits and releases its locks
($c_3$ : doesn't happen yet because T3 is delayed)
11: $c_4$ : T4 commits and releases its locks; T3 is re-activated
12: $c_3$ : T3 commits and releases its locks

So, the order of execution (the schedule) is
$r_1(x)\ r_1(y)\ w_1(x)\ r_4(y)\ c_1\ w_2(x)\ c_2\ c_4\ w_3(y)\ c_3$

e) i) Dirty read: the query reads attribute validated for an entry record which is written by the update
(Phantom: based on the definition of the table you could argue that the update may move a record from one group to the other for the query).

ii) In case of only 'Dirty read' as anomaly:
Update: READ UNCOMMITTED
Query: READ COMMITTED
In case of both 'Dirty read' and 'Phantom' as anomaly
Update: READ UNCOMMITTED
Query: SERIALIZABLE

f) A constraint that makes sure a referenced record is always present
(An index on eid for the table entry)

## Question 3

Question 1. JAXB facilitates the development of RESTful services in Java by performing all the XML serialisation and parsing for the programmer. The programmer only needs to define a Java Bean class and annotate that as @XMLRootElement. Whenever the code refers to an instance (object) of this Java Bean, XML serialisation and parsing is done automatically.

Question 2.

a. The HTTP request to refer to the URL http://somedomain/students/S234567, HTTP method should be PUT or POST and the HTTP message body should contain a representation of the student according to the resource representation supported by the web service.

b. There should be an agreement between the client and the web service on the representation format, otherwise they cannot understand each other. This determines how the resources (students) are represented in the messages. The most popular representations are JSON representation and XML. For example, an XML representation of this student could be:

```
<student>
  <sutdentid>S234567</studentId>
  <fullName>Peter Perfect</fullName>
   <courses>
    <course>
      <courseId>201300180</courseId>
      <name>Data and Information</name>
    </course>
  </courses>
</student>
```