Data & Information – Test 3 (1.5 hours)

8 June 2018, 13:45–15:15

Program: Technical Computer Science / Business & IT
Module: Data & Information (201700279)
Module Coordinator: Klaas Sikkel
Responsible Teachers: Maurice van Keulen / Luís Ferreira Pires

Please note:
- Please answer every question on a different sheet of paper (the answers will be distributed to different person for grading).
- You are not allowed to bring any study materials to the test; essential excerpts from the study materials are available as appendices.
- You are allowed to use a calculator.

Grade = #points/10

Question 1: Information Retrieval / Full-text Search (35 points)

Tip: See Appendix 1 for an informal syntax of SQL including types, functions and operators important for full-text search.

a) [FT-search] Suppose we have the tables below. The table 'person' contains both students and teachers and the table 'project' contains student project descriptions as well as which student the project belongs to and which teacher is supervising the project.

Write an SQL query that uses the full-text search capabilities for finding all projects that have a description that matches "databases | (data & science)".

```
CREATE TABLE person (
    nr INTEGER,
    name TEXT,
    PRIMARY KEY (nr)
)
```

```
CREATE TABLE project (
        pid INTEGER,
        student INTEGER,
        supervisor INTEGER,
        description TEXT,
        PRIMARY KEY (pid),
        FOREIGN KEY (student) REFERENCES person(nr)
        FOREIGN KEY (supervisor) REFERENCES person(nr)
)
```

b) [FT-search] Given the tables above. Write an SQL query that uses the full-text search capabilities for finding projects that match a search keyword *K* in either the description or the name of the student or the supervisor.

c) Which of the following statements is true; explain your answers

   i. Converting a string to a tsvector leaves the words intact.

   ii. When converting a string to a tsvector, every word in that string becomes part of the resulting tsvector.

   iii. There exists an index structure capable of speeding up full-text queries.

d) [IR; tf.idf] Given a table with quotes as illustrated below (filled with quotes from https://www.coolfunnyquotes.com), calculate the tf.idf scores for the query 'why friend'. Explain your answer by giving the full calculation. The formulas for ranking and idf are given below the table. If you don't have a calculator for computing log, just invent a number between 0 and 1 (explicitly mention that you did this!). The terms of the query are given in bold for your convenience (we assume 'stemming' so the words "friends" and "friend" are both considered the same term "friend")

Qid	**Author**	**Quote**
1	Anonymous	If I won the award for laziness, I would send somebody to pick it up for me.
2	Anonymous	Maybe if we tell people the brain is an app, they'll start using it.
3	Anonymous	There's nothing better than a good **friend**, except for a good **friend** with chocolate
4	Anonymous	If Cinderella's shoe fit perfectly, then **why** did it fall off?
5	Anonymous	A best **friend** is like a four-leaf clover, hard to find, lucky to have.
6	Anonymous	If we shouldn't eat at night, **why** is there a light in the fridge?
7	Anonymous	If you're hotter than me, then that means I'm cooler than you.
8	Anonymous	Never let your best **friends** get lonely... keep disturbing them.

Idf(t) = log (N/df)

Rank(d,q) = $\sum_{t \in q}$ tf(d,t) idf(t)

e) [IR; language models] In language models, it is often assumed that terms are independent (this is called the unigram model). In the bigram model, probabilities P(w1 w2|D) are determined and stored for all combinations of w1/w2, so that we need not assume independence P(w1 w2|D) = P(w1 | D) * P(w2 | D). Can you give a typical example from the quotes above, where P(w1 w2|D) >> P(w1 | D) * P (w2 | D)?

Question 2: Database Transactions, Isolation Levels, Indices, Constraints, Views (35 points)

Tip: See Appendix 1 for an informal syntax of SQL.
Tip: See Appendix 2 for a table (the same table as in the slides of lecture DB-5) which summarizes the SQL isolation levels, the anomalies they prevent, and the locking implementation of those isolation levels.

a) Given the schedule below, which pairs of operations are conflicting?

$r_1(x)\ r_1(y)\ r_2(x)\ r_3(y)\ w_1(x)\ w_2(x)\ w_3(y)\ c_1\ c_2\ c_3$

b) Is the schedule above serializable or not? Explain your answer

c) Suppose the database would run under isolation level SERIALIZABLE, i.e., it would use long-term read- and write-locks, what schedule would emerge for the above stream of operations? Explain your answer.

d) Suppose there are only two applications running on the same database accessing the same single table contained in this database. The first application is a read-only application where each SELECT-statement is a single transaction. The other application only appends rows to this table in batches, i.e., several rows in one transaction. For each anomaly, explain why it can or cannot occur in this situation.

e) Suppose a table “project” is created containing a declaration “FOREIGN KEY (student) REFERENCES person(nr)” and filled with data of some projects.

   i. Describe what happens when we issue a “DELETE person” statement deleting all rows in the person-table.

   ii. Describe what happens when we issue two statements “DELETE person” and “DELETE project” in the same transaction in an attempt to delete all data in the database.

   Explain your answers.

Question 3: REST (30 points)

a) How does a RESTful client indicate to the server the encoding to be used in the data contained in the response to a certain HTTP request? How does a RESTful server implemented using Jersey (JAX-RS) select the Java method that gives a response with the proper encoding?

b) Suppose a web service is available that manages a collection of students in an administrative application of a university. Assume that the REST URL conventions have been properly followed so that
   - `http://anyhost/students` represents the whole collection of students
   - `http://anyhost/students/`*`id`* represents a student with identifier *id*

   Explain how a client can *update a student* (Update REST action) by describing the HTTP messages (request/response) that are exchanged to do this and the possible contents of these messages (HTTP method, URL and message body). Discuss both the successful and unsuccessful executions!

c) The following statements have been added to the `/WEB-INF/web.xml` file in a project that implements a RESTful service using JAX-RS:

```
<servlet>
      <servlet-name>javax.ws.rs.core.Application</servlet-name>
</servlet>
<servlet-mapping>
      <servlet-name>javax.ws.rs.core.Application</servlet-name>
      <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

   What has been defined in these statements? How does the Application Server (Tomcat) interpret this definition in order to eventually reach the intended RESTful service?

Supporting information

JAX-RS annotations

```
@Path(“somePath”)
@GET, @POST, @PUT, @DELETE
@Produces(“...”)
@Consumes(“...”)
@*Param(“...”)
```

REST action vs. HTTP method mapping

REST action	HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Appendix 1: Informal syntax of SQL

In the informal syntax, we use the following notations

• A | B	to indicate a choice between A and B
• [A]	to indicate that A is optional
• A*	to indicate that A appears 0 or more times
• A+	to indicate that A appears 1 or more times
• 'A'	to indicate that the symbol A is literally that symbol

We are not precise in punctuation in the syntax, but this is irrelevant in this exam anyway.

SQL

createtable: CREATE TABLE tablename '(' columndef+ constraint* ')'
createview: CREATE VIEW viewname AS query
query: SELECT (column [AS colname])+ FROM (tablename [AS colname])+ WHERE condition
  [GROUP BY column+] [ORDER BY column+]

columndef: colname type [NOT NULL] [UNIQUE] [PRIMARY KEY] [REFERENCES tablename (colname+)]
constraint: PRIMARY KEY (colname, ...)
  | FOREIGN KEY(colname, ...) REFERENCES tablename(colname, ...) | CHECK (condition)
column: [tablename '.'] colname | '*'

Examples of condition:
  column = value [(OR | AND) [NOT] column <> value]
  | column IS [NOT] NULL
  | column [NOT] IN (value, ...)
  ...
Full-text types: tsvector, tsquery
Full-text functions: to_tsvector, to_tsquery, ts_rank, ts_headline, string_agg, setweight, coalesce
Important operators: @@ (match), || (concatenate), :: (cast)

Appendix 2: Isolation levels

isolation level	**prohibited anomalies**	**definition anomaly**	**implementation** prohibiting the anomalies
READ UNCOMMITTED	dirty write	… $w_2(x)$ … $w_1(x)$ …	only write locks
READ COMMITTED	dirty read	… $w_2(x)$ … $r_1(x)$ …	short-term read locks
REPEATABLE READ	non-repeatable read	$r_1(x)$ … $w_2(x)$ … c_2 … $r_1(x)$ … c_1	Long-term read locks
SERIALIZABLE	phantom	$r_1(P)$ or $w_1(P)$ … $w_2(y \text{ in } P)$	Long-term predicate locks