Modern Database Systems
Assignment 2
Due by **14 March 2016**

Note 1 Use mycourses.aalto.fi to submit your solutions. The name of the file you submit should have the following format.

modernDB2 – [your-student-id] – [your-full-name].pdf

Moreover, make sure that the cover (first page) of your pdf contains the following pieces of information: (i) first name, (ii) surname, (iii) student id, and (iv) email.

Note 2 This assignment is personal and no solutions should be shared. If you have discussed the tasks with someone else, please mention their name in your submission.

Note 3 The grading scheme is as follows.

	Part A	Part B	Total
Max Points	40	40	80

Points of individual questions are mentioned next to the question numbers.

PART A: Pen and Paper

Problem 1 (20 points)

We want to design a web search engine. We will need to index a collection of documents (web pages) $D = \{d_1, \ldots, d_n\}$. Each document $d \in D$ is represented as a set of terms and their multiplicities, i.e., $d = \{(t_1, f_i), \ldots, (t_m, f_m)\}$ indicating that term t_i appears f_i times in document d. A query q is also represented as a set of terms (keywords), $q = \{k_1, \ldots, k_\ell\}$. Given a query q and a document $d \in D$ we define the *relevance score* $r(d,q)$ of the document d with respect to the query q to be

$$r(d,q) = \begin{cases} \frac{1}{2}\, i(d) + \frac{1}{2} \cos(d,q) & \text{if } q \text{ is contained in } d, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

where

$i(d)$: is a *query-independent* term that intends to capture the importance of document d. We assume that $i(d)$ takes values between 0 and 1, and the larger the value the most important is the document.

$\cos(d,q)$: is the cosine similarity of d and q, when those are viewed as vectors. Note also that for computing the cosine similarity the terms of d are weighted by a `tf.idf` coefficient.

Given a query q, we need to return the documents in D ranked by relevance score $r(d,q)$. To make the computation efficient we will use an *inverted index* $\mathscr{I}$, as discussed in class.

Question 1: Propose two different definitions for the function $i(d)$. (This is an open-ended question and you can be as creative as you want.)

Question 2: Describe in detail what information you need to keep in the inverted index $\mathscr{I}$ in order to be able to efficiently compute the ranked list of relevant document for each query q.

Question 3: As it is common in most web searches, a user sees the top-10 results in the first page and very rarely clicks to see the results in the second page. This means that we only need to compute the first 10 results, and compute the next 10 *only* if a user clicks 'next page'.

Describe how you will organize the inverted index $\mathscr{I}$ in order to be able to compute efficiently the top 10 results. Describe your algorithm in detail. Note however, that your algorithm needs to compute the correct answer, that is, the 10 documents with the highest relevance score, not just any 10 matching documents.

Question 4: Assume now that in order to speed up the computation we allow to return approximate answers. In particular, for a given parameter $\epsilon > 0$, we want to return ϵ-approximation of the top-10 documents.

Propose a definition of what it means for a set of 10 documents to be an ϵ-approximation top-10.

Describe how to modify your algorithm in Question 3 in order to handle ϵ-approximation top-10 results.

Discuss why this modified algorithm is more efficient than the algorithm in Question 3.

Problem 2 (20 points)

Question 1: Prove the correctness of the threshold algorithm (TA) for each of the following cases:

(1) Restricted random access;

(2) Approximate top-k.

Question 2: Provide an example (a dataset with n objects and an aggregation function) where the TA algorithm will make $\mathcal{O}(n)$ accesses (both sorted and random accesses) to return a top-k answer, while an algorithm that makes wild guesses will return the correct answer with $\mathcal{O}(1)$ accesses.

PART B: Programming

Setup

You are provided with:

- a VirtualBox[1] instance running Ubuntu, with Python 3.5 and MongoDB 3.2, PyMongo 3.2.1, and Jupyter installed,
- three collections of json/bson documents, already imported into a MongoDB database.

As with the previous assignment, we have setup a VirtualBox (VB) instance with the required resources. You can obtain it by logging in to a niksula machine (Paniikki lab), and issuing the following command at a terminal.

> cp /scratch/mathiom1/moderndb-ubuntu.ova /scratch/your-username/

When you use VirtualBox to mount the instance (choosing *"Import Appliance"* from the menu), make sure to set the

hard disk controller (SATA) / virtual box image

attribute to point to **your scratch directory**.

hard disk controller (SATA) /virtual box image: /scratch/your-username

IMPORTANT!
When you use the Panikki Lab, do NOT store any
of the provided files under your home folder.
Instead, use the space under the /scratch/your-username directory.

If you prefer to open the VirtualBox instance on a different computer, please use the Paniikki Lab to access the dataset as described earlier.

The VirtualBox instance comes with Ubuntu and you can login with the following credentials.
username: **student**
password: **moderndb**

Before you test your queries, make sure that the mongodb daemon (program *mongod*) is running. You can do that by simply issuing the command

> mongod

in a terminal.

Data

You are provided with three collections, *homeSales.bson*, *postCodes.bson*, and *companies.json* under the following directory.

/home/student/moderndb/assignment2

The collections have already been imported to mongodb with the following commands.
> mongorestore --db moderndb --drop homeSales.bson
> mongorestore --db moderndb --drop postCodes.bson
> mongoimport --db moderndb --collection companies --drop --file companies.json

[1] https://www.virtualbox.org/

Problem 3 (25 points)

This question concerns collections *homesSales* and *postCodes*. Each document in *homeSales* contains information about a single home sale and each document in *postCodes* contains information about a single postcode (a.k.a. postal code).

For each of the following questions, provide a single mongodb query to extract the information necessary to answer the question.

3.A

(1) How many home sales are there in collection *homeSales*?
(2) Some documents in collection *homeSales* are associated with an "amount" field. What is the maximum "amount" value in the collection?
(3) Let M be the maximum amount value you found in the previous question. How many home sales in the data are more expensive than $0.5M$? (You do not have to compute M again for this query -- just use directly the value you found from the previous query).
(4) How many home sales are more expensive than $0.1M$ but less expensive than $0.5M$?
(5) Find all home sales from postcode "SL6 4LG".
(6) Some documents in collection *homeSales* are associated with a "date" field. What is the minimum and maximum "date" in the collection?
(7) How many distinct "postcode" values so appear in the data?
(8) How many home sales are there for each postcode?
(9) Considering only home sales in town "MAIDENHEAD", how many home sales are there for each postcode in that town?
(10) Considering only home sales in town "MAIDENHEAD", how many home sales were there for each of the top 3 postcodes in that town? Write a query that returns only these top-3 postcodes in terms of number of sales and the number of sales they had.
(11) Considering only home sales in town "MAIDENHEAD", what was the average amount for each postcode in that town?
(12) Among all postcodes that had more than 5 sales in the data, what is the postcode with highest average sale amount? Write a query that returns this postcode and its average amount.
(13) Find all home sales for county "BUCKINGHAMSHIRE" and save them in a new collection named "mycounty".
(14) Each postcode is associated with a pair of coordinates that mark the center of its region (see collection "postcodes"). For each sale in collection "mycounty", find the amount of the sale and the coordinates of its postcode. Write a query that returns only that and no additional information.
(15) Consider the sales represented in collection "mycounty" and particularly addresses for which there is more than one sale. Find the address with the maximum absolute difference of the amount between two sales. Write a query to return this address and amount difference and nothing else.

3.B

1. Provide a relational schema to accommodate the same information that is stored in collections *homeSales* and *postCodes*. Explain your answer.
2. For the relational schema you defined, provide the SQL queries for questions **3.A** (2, 4, 8, 9, 10).

Problem 4 (15 points)

For this question, we consider collection *companies*. The collection contains information about start-ups, including their names (field "*name*"), the year when they were founded (field "*founded_year*"), and their rounds of funding. Regarding the latter, the collection contains information about the year each when funding round took place and the amount of money raised in US dollars (fields "*funding_rounds.funded_year*" and "*funding_rounds.raised_amount*", respectively).

For each of the cases below, you are asked to write a query in MongoDB to extract some information from the collection and also select an appropriate index to support the query.

- In all cases, you should write the query using the **find()** function.
- Do ***not*** use the aggregate() function for this question.
- In all cases, you can assume that all amounts of money in the collection that are related to *funding_rounds* are expressed in US dollars.

Case 1

We are interested in finding the **names of companies that raised more than 5,000,000 US dollars in a round**.

a. Write a query in MongoDB to extract this information.

b. Consider the following two indexes.

I. **index** built on field **funding_rounds.raised_amount** in **ascending** order,
II. **index** built on field **name** in **descending** order.

Which of the two indexes (if any) does MongoDB use to execute the query you provided in part (a)? Why? If MongoDB does use either of the two aforementioned indexes, how much faster does your query run compared to when neither index is built?

Case 2

We are interested in finding the **names of companies that raised more than 5,000,000 US dollars in a round that took place before 2005.**

a. Write a query in MongoDB to extract this information.

b. Consider the following two indexes.

I. **index** built on field **funding_rounds.raised_amount** in **ascending** order,
II. **compound index** built on: first, field **founded_year** in **ascending** order, and second, on field **funding_rounds.funded_year** in **ascending** order.

Which of the two indexes (if any) does MongoDB use to execute the query you provided in part (a)? Why? If MongoDB does use either of the two aforementioned indexes, how much faster does your query run compared to when neither index is built?

Case 3

We are interested in finding **the names of companies that were founded before 2000**.

a. Write a query in MongoDB to extract this information.

b. Consider the following two indexes.

I. **compound index** built on: first, field **founded_year** in **ascending** order, and second, on field **funding_rounds.funded_year** in **ascending** order.

II. **compound index** built on: first, field **funding_rounds.funded_year** in **descending** order, and second, on field **founded_year** in **descending** order.

Which of the two indexes (if any) does MongoDB use to execute the query you provided in part (a)? Why? If MongoDB does use either of the two aforementioned indexes, how much faster does your query run compared to when neither index is built?