Modern Database Systems
Assignment 3
Due by **15 April 2016**

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

modernDB3 – [your-student-id] – [your-full-name].[pdf/ipynb]

For this assignment, we expect you to submit two files – one for the pen-and-paper part (pdf), and one for the programming part (IPython notebook, .ipynb).

Moreover, make sure that the first page of all files you submit contain 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	50	50	100

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

PART A: Pen and Paper

Problem 1 (10 points)

Design a map-reduce algorithm to perform matrix multiplication in one map-reduce round.

Problem 2 (20 points)

We discussed in class that the bottleneck of a map-reduce program is the amount of data that is moved between the different map and reduce tasks. Thus, it is reasonable to measure the efficiency of a map-reduce program using the notion of *communication cost*, measured in the following way:

0. First, note that a map-reduce program is a sequence of tasks, which can be either map or reduce tasks.

1. The communication cost of a task, either map or reduce, is the size of the input data that is fed to the task.

2. The communication cost of a map-reduce program is the sum of the communication costs of all of its tasks.

In the questions below we are considering operations over relational tables. In this case we are measuring the input size of a task as the number of tuples that is fed in a task.

Question 2.1. What is the communication cost of the map-reduce algorithm for computing the join $R(A,B) \bowtie S(B,C)$?

The algorithm is given in slide 14 of lecture 9.

We now want to compute a 3-way join $R(A,B) \bowtie S(B,C) \bowtie T(C,D)$. We consider two different ways to compute this 3-way join:

(L) $(R(A,B) \bowtie S(B,C)) \bowtie T(C,D)$, and

(R) $R(A,B) \bowtie (S(B,C) \bowtie T(C,D))$.

Question 2.2. Express the communication cost of alternatives **(L)** and **(R)** and devise conditions when one is preferable than the other.

Problem 3 (20 points)

Consider the problem of computing *similarity self-join* that we saw in class: we are given a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of n unit vectors in $\mathbb{R}^d$ and a similarity threshold θ and the goal is to find all pairs of vectors in X whose dot-product similarity $\text{sim}(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$ is at least θ.

Consider the *prefix filtering* method as described in the slides: we first define a *max vector* $\overline{\mathbf{x}}$ whose j-th coordinate is $\overline{x}_j = \max_{\mathbf{x} \in X} x_j$. Then for each vector $\mathbf{x} \in X$ we define $p(\mathbf{x})$ to be the *largest integer* such that

$$\sum_{j \leq p(\mathbf{x})} x_j \overline{x}_j < \theta$$

We define the *signature* $S(\mathbf{x})$ of vector $\mathbf{x}$ to be the set of coordinates after $p(\mathbf{x})$, i.e.,

$$S(\mathbf{x}) = \{(j, x_j) \mid j \geq p(\mathbf{x})\}$$

Question 3.1. Prove that if the signatures of two vectors $\mathbf{x}, \mathbf{y}$ do not intersect then $\text{sim}(\mathbf{x}, \mathbf{y}) < \theta$.

Question 3.2. Discuss how to use the property proven in Question 2.1. in order to compute the similarity self-join of a set of vectors efficiently.

PART B: Programming

For this part, you'll be using PySpark to process *tweets*, i.e., messages generated on *twitter*.

Specifically, you will be using the IPython notebook we have created and you will run it on a server provided by CSC (www.csc.fi). The notebook is posted on *mycourses* and contains setup instructions. When you submit your solutions to *mycourses*, you should also submit your copy of the notebook (code and results).

You will work with two datasets:
· A **small** dataset. This dataset is already available. Its purpose is for you to work with it to develop solutions for the assignment.
· A **large** dataset. This dataset will be available on *Friday April 1st, 2016*, together with instructions on how to access it. You will use it to produce your *final solutions* for the assignment.

In what follows, we provide the problems for the programming part.
For complete instructions, see the notebook.

Problem 4 (20 points)

Use PySpark to write and execute queries for the following tasks (1 - 10).

Tasks
1. Find out how many lines (tweets or corrupt) there are there in the data.
2. Inspect any three tweets and infer the tweet schema from them, as completely as you can.
3. Each tweet is identified by its unique *'id'* value. Some tweets might appear in more than one lines in the data (i.e., the same tweet *id* might appear in more than one lines). How many tweets are there that appear in *only* one line?
4. Each tweet is associated with a '*lang'* value, that denotes the language the tweet was written in. How many different languages are there in the dataset?
5. How many *lines* are there in the data for each language? (Ignore tweet ids for this task).
6. How many *tweets* are there in the data for the english language (*lang = 'en'*)?
7. Each tweet is associated with a text value that stores the content of the message. What is the minimum and maximum message length in the data? *Note*: you should compute both values in a single pass over the data.
8. Consider the text message that appears in a tweet. We define a *word* to be a maximal sequence of alphanumeric characters found in the text, after the text has been converted to *lowercase*. For example, if the text is

My username is spark123 & my password is spar!.Kk

then the words contained in it are the following.

my, username, is, spark123, my, password, is, spar, kk

Find the 1000 most frequent words in the text messages of all english tweets (*lang = 'en'*) along with the number of their occurances.
9. Find the 100 most frequent words in english tweets (*lang = 'en'*) that start with each character of the latin alphabet ('a', 'b', ..., 'z'). Use the *lowercase* version of messages.
10. Each tweet is associated with a user who generated it; and the user is associated with a unique '*screen_name'*. Find the screen_name of the 10 users with the most english tweets (with distinct tweet ids) in the data.

Note Twitter users are also associated with an '*id'* value, different than the tweet '*id'*. Make sure you do not confuse the two.

Problem 5 (30 points)

Some tweets are *replies* to tweets written by other Twitter users.

You can tell which tweets are *replies* by checking the '*in_reply_to_screen_name*' value of a tweet: if it is *None*, then it is not a reply - otherwise, it is a reply to the user with the '*screen_name*' mentioned therein.

For example, *in_reply_to_screen_name: None* signifies that the tweet is not a reply to another tweet, but *in_reply_to_screen_name: northern_bytes* signifies that the tweet is a reply to another tweet generated by user *northern_bytes*.

We are going to construct a graph in the following steps:
1. We place one node in the graph for each user who has produced more than 20 english tweets (with distinct ids) in the data;
2. We place one directed edge from node *u* to node *v* iff the total number of english replies from user *u* to user *v* in the data is more than 10. In that case, we will say that '*u* is connected to *v*'.

1. Graph construction

Construct a pair RDD named **linksRDD** to store the adjacency list of each node. Specifically, for each node (user) *u* in the graph, you should have one element of the following form

(screen_name, [screen_name_1, screen_name_2, ..., screen_name_v, ...])

where *screen_name* corresponds to user *u* and *screen_name_v* corresponds to a user *v* that *u* is connected to.

How many nodes and edges are there in the graph?

2. Pagerank Computation

Implement and employ PageRank on the graph you constructed above, using 10 iterations and parameter α=0.15. Store the pagerank scores in an RDD named **ranksRDD**.

Report the 5 nodes with highest pagerank values.