


Family name: \_\_\_\_\_ Given name: \_\_\_\_\_

- 1) (20%) In the framework of the **RUM conjecture**, name six data structures we saw in the course (not concrete tool implementations) and place them in the corresponding category:

Read optimized: \_\_\_\_\_ 

Write optimized: \_\_\_\_\_

Space optimized: \_\_\_\_\_


- 2) (15%) Which is the main problem in having **replicas**, and which is the innovation introduced by some NOSQL tools to solve it.

Problem: \_\_\_\_\_

Innovation: \_\_\_\_\_ 

\_\_\_\_\_

- 3) (15%) Suppose you have a hash function whose range has size 100 (i.e.,  $D=100$ ), and a **Consistent Hash** structure with 5 machines ( $M1..5$ ) whose identifiers map to values  $h(M1)=0$ ,  $h(M2)=20$ ,  $h(M3)=40$ ,  $h(M4)=60$ ,  $h(M5)=80$ . What happens if you have an object mapped to value  $h(O)=90$ ?

\_\_\_\_\_ 

- 4) (25%) Suppose you implement a system to store images in hundreds of machines with thousands of users using **HBase** with a single column-family. These images taken at time  $VT$  belong to a person  $P$  who tags each with a single subject  $S$  (e.g., family, friends, etc.) and are concurrently uploaded into the system at time  $TT$  in personal batches containing multiple pictures of different subjects taken at different times. Each person can then retrieve all his/her pictures of one single subject that were taken after a given time. Precisely define the key you would use if you exclusively prioritize (i.e., do not consider any other criteria) ...

... Load balancing on ingestion: \_\_\_\_\_

Assumptions made: \_\_\_\_\_

\_\_\_\_\_

... Load balancing on querying: \_\_\_\_\_

Assumptions made: \_\_\_\_\_

\_\_\_\_\_

... I/O on ingestion: \_\_\_\_\_ 

Assumptions made: \_\_\_\_\_

\_\_\_\_\_

... I/O on querying: \_\_\_\_\_

Assumptions made: \_\_\_\_\_

\_\_\_\_\_

5) (25%) Consider two files containing the following kinds of data:

Employees.txt

EMP1;RICARDO;250000€;MADRID;SITE2  
EMP2;EULALIA;150000€;BARCELONA;SITE1  
EMP3;MIQUEL;125000€;BADALONA;SITE3  
EMP4;MARIA;175000€;MADRID;SITE2  
EMP5;ESTEBAN;150000€;MADRID;SITE4

Departments.txt

SITE1;DPT.MANAGEMENT;FLOOR10;ST.PAU CLARIS;BARCELONA  
SITE2;DPT.MANAGEMENT;FLOOR8;ST.RIOS ROSAS;MADRID  
SITE3;DPT.MARKETING;FLOOR1;ST.PAU CLARIS;BARCELONA  
SITE4;DPT.MARKETING;FLOOR1;ST.RIOS ROSAS;MADRID  
SITE5;DPT.MARKETING;FLOOR5;ST.MARTI PUJOL;BADALONA

Provide the ordered list of **Spark operations** (no need to follow the exact syntax, but just the kind of operation and main parameters) you'd need to retrieve the list of department IDs for those departments with sites in all cities where employees live (these employees can be even from other departments). Save the results in "output.txt". In the previous example, the result should be "DPT.MARKETING", because it has sites in all the three cities where there are employees (i.e., MADRID, BARCELONA and BADALONA). However, "DPT.MANAGEMENT" should not be in the result, because it does not have any site in BADALONA, where EMP3 lives.

- 1) \_\_\_\_\_
- 2) \_\_\_\_\_
- 3) \_\_\_\_\_
- 4) \_\_\_\_\_
- 5) \_\_\_\_\_
- 6) \_\_\_\_\_
- 7) \_\_\_\_\_
- 8) \_\_\_\_\_
- 9) \_\_\_\_\_
- 10) \_\_\_\_\_
- 11) \_\_\_\_\_
- 12) \_\_\_\_\_
- 13) \_\_\_\_\_
- 14) \_\_\_\_\_
- 15) \_\_\_\_\_