Homework 1: Distributed RDBMS and NoSQL

Form:	Theoretical - Written report (PDF) Practical – Jupyter notebook
Language:	English or Hebrew
Requirements:	Theoretical - The report should be up to 2 pages Practical - The code should run without errors
Submission:	Course Website – moodle
Contact:	tamirmendel@mail.tau.ac.il
Deadline for submission:	**20.5.2019**

Students will form teams of three people each and submit a single homework for each team. The same score for the homework will be given to each member of the team. Please write all group members ids in the document and Jupyter notebook.

The goal of this homework is to test your understanding of the concepts presented in the first three lectures and the first lab. If your answers are based on a material you have read elsewhere, please make sure to avoid plagiarism (i.e., cite the source and phrase each answer with your own words).

Theoretical

Question 1: Describe the four ACID properties and for each of them provide an example that shows how the property can be broken when no mechanism is used to ensure it.

Question 2: Describe replication and fragmentation in the context of a distributed RDBMS. Provide some of the advantages and disadvantages of both. Describe two real-world examples representing the benefits in using vertical and horizontal fragmentation.

Question 3: Read the paper: “Bigtable: A Distributed Storage System for Structured Data” and describe the main advantages and limitations of the Big Table platform (your answer should not exceed half a page!).

Practical

The homework assignment should be submitted in a Jupyter notebook (including outputs). Use Markdown or comments for the important code lines. The Jupyter notebook should run without errors or bugs by order top to bottom.

Scenario: Your are asked to develop a website that will allow companies to post jobs and candidates to apply for a job position. For simplicity, you can assume that the generated data has the following structure:

- Candidate: Candidate Name, Email, LinkedIn URL, Skills List
- Jobs: Jobs ID, Job Name, Requirements List, Location, Publish Date, Status, Candidates List
- Company: Company Name, Company Description, Jobs List

The system will comprise two components: persistent storage and an in-memory cache. The persistent storage will be implemented using MongoDB and will contain all the information that must be stored for the correct functioning of the website, as well as for analytical purposes. Note that for the purposes of the exercise you are not allowed to use more than **one collection**. The in-memory cache will be implemented using Redis and will take care of the operational side including fast operations.

You are asked to define and implement functions to support the following operations:

Operation 1: Add a new company. Define and implement the function *add_company(company)* that adds a *company* (a python dictionary) to the system. Assume that the Company Name is unique, so you cannot accept more than one Company with the same Company Name. An example of a call to this function is as follows:

```
add_company({'company_name':'TAU', 'company_description':'University'})
```

Operation 2: Add a new job position. Define and implement the function *add_job(job, company_name)* that adds a *Job* (a python dictionary) to a Company. The Job Id should be unique for a given company, and should be generated automatically when a new Job is added to a company. An example of a call to this function:

```
add_job({'job_name':'bi developer', 'location': 'Tel Aviv',
'skills':['python','big data','mongodb'],'status':'open','publish_date':'01-
02-2019'},'TAU')
```

Operation 3: Define and implement the function *application(candidate, application_time, job_id)* that adds a *Candidate* to a Job. Assume that the emails of candidates applying to the same job should be unique (i.e., an application of a candidate to a job that was already applied by a candidate with the same email should not be allowed). Note that the application should also include the application_time:

```
application({'candidate_name':'laura', 'email':'laura@gmail.com',
'linkedin':'https://www.linkedin.com/in/laura/', skills: ['python','sql']},
'01-02-2019 15:00:00', '1') #note that this means that Laura applied for job
number 1
```

Operation 4: Given a candidate, show the latest K companies that the candidate applied to. Define and implement the function *show_latest_applications(k, candidate_email)*. An example of a call to this function:

```
show_latest_apply(10, 'laura@gmail.com')
```

Operation 5: Show candidates' emails that applied for a given position and have the specified skills. Define and implement the function *show_candidates(job_id, skills)*. An example of a call to this function:

```
show_candidates('1', ['python','sql'])
```

*Note: assume that the skill 'Y' is included in the job requirements list.

You are also asked to define and implement functions to produce the following reports:

Report 1: Define and implement the function *count_jobs_by_company()* that count how many jobs have been posted for each company name.

Report 2: Define and implement the function *count_candidates_by_job()* that count how many candidates have for each Job ID during the last 30 days.

Recovery

Create the function *recovery()* that should be executed whenever the Redis server restarts (for example after a server crash like power failure). This function will use the persistent data (stored on MongoDB) to reload data to the cache (Redis).

Execution

Create the function *execute()* that call all functions with the arguments that are given in the examples. All the function should return an output. Print all the outputs in a readable and nice displayed.

Evaluation

Special attention will be given to the data model that you defined and to the efficient implementation of the different operations/reports, so you should make sure to include in the notebook also a description of your model and motivations of your implementation choices. The code should be organized and the outputs should be nice displayed.

Good Luck!