HW1 Big Data Management

By Maëlys Boudier & Natalia Beltrán

General Remarks:

- Open and run the Jupyter Notebook file to run all the models and query
- If you wish to change the number of documents, change employee_count in the data_generator() functions; currently set at employee_count = 50000

Table 1: Query Execution Times per Model (*in seconds*)

	Q1	**Q2**	**Q3**	**Q4**
M1	**2.2888e-05**	0.0952	**0.2225**	**0.00697**
M2	1.9789e-05	0.0536	0.2338	0.3825
M3	2.8133e-05	**0.0526**	0.5434	0.1809

Answer the following questions (make sure you justify your answers and not only list them):

1. Order models from best to worst for **Q1**. Which model performs **best**? Why?
Order: M2, M1, M3.
Model 2 performs the **best**, likely because it follows a denormalized data structure where "Company" is embedded within the "Person" document. This means that when querying for a person's full name and their company's name, the database engine can fetch all the required information from a single document, resulting in fewer disk reads and potentially faster retrieval times compared to the other models.

2. Order models from best to worst for **Q2**. Which model performs **best**? Why?
Order: M3, M2, M1.
Model 3 performs the **best** since it accesses each company document and sums the number of embedded documents (which each correspond to employees). It can simply output the company name from the main document and the number of employees it aggregated. Model 2 is a little bit slower but is a close second and takes a different approach which has to iterate through every employee document to aggregate the sum for each company. The embeddings make this quicker than Model 1 which has 2 separate document classes.

3. Order models from worst to best for **Q3**. Which model performs **worst**? Why?
Order: M3, M2, M1.
Model 3 performs the **worst** because to access the employee's birthdate and age, we need to access every company and then all the embedded employee documents within it. This would take more time then Model 2 which could directly access the employee's personal information in the

first level. We structured query 3 to slice the first four characters of the birth date (in isoformat), convert it to an integer, and compare it with the year 1988 setting the age of those born before 1988 to 30. The denormalized structures are much slower than the normalized structure (Model 1) at updating information.

4. Order models from worst to best for **Q4**. Which model performs **worst**? Why?
Order: M2, M3, M1.
Model 2 performs the **worst** because to access each company name it needs to first access all of the employees and then access the embedded company documents one level below to update the name to include the word "Company". This would take longer than accessing the company name in Model 3 in the main document. The denormalized structures are once again much slower than the normalized structure (Model 1) at updating information.

5. What are your conclusions about **denormalization or normalization** of data in MongoDB? In **the case of updates,** which offers better performance?
The speed of data access and updates heavily depends on the structure of the embeddings and documents. Therefore, it's crucial to consider the model structure based on which data needs to be accessed or updated most frequently. Data that requires frequent access should be placed on the first level, while less frequently updated data can be stored in embedded documents. For instance, Model 2 experienced slower updates in query 4 because it had to traverse through embedded documents, unlike the other models which accessed data more directly. Model 1 had generally slower read times when accessing data across both documents, but had much faster update times.

Denormalization tends to excel when read performance is important, as it optimizes data reading efficiency. On the other hand, in the case of updates, normalization offers better performance is important, as it provides efficient data representation for updates.