Barcelona School of Economics

Assignment 2

Big Data Management - L2-T01

Lucia Sauer

Julian Romero

May 26, 2025

C. Results and Discussion

To ensure reliable performance results, each query was run five times and the average execution time computed, reducing the impact of caching and other transient system effects. This provides a stable measure of typical performance. The three data models were generated with 500K companies and 5M persons, simulating realistic workloads. The results assess query efficiency by execution time. Table 1 reports the average time per query and model, while Table 2 shows results relative to the fastest execution time for each query.

	Q1	**Q2**	**Q3**	**Q4**
M1	48.2893	4.3074	8.9651	2.0975
M2	5.6436	2.2589	9.3233	30.1073
M3	5.2151	0.6193	2.7609	3.7778

Table 1: Execution times (s) by model

	Q1	**Q2**	**Q3**	**Q4**
M1	x9	x7	x3	x1
M2	x1	x4	x3	x14
M3	x1	x1	x1	x2

Table 2: Execution relative to the fastest

Q1 *M3≈M2>M1*

Although M2 only needs to project the `fullName` and `company.name` without any transformation, it must scan all 5,000,000 person documents, which impacts performance. In contrast, M3 stores persons embedded within company documents and requires an `unwind` operation on the `staff` array to access each person's `fullName` before projection. While this adds one extra operation compared to M2, it processes significantly fewer documents (500,000), making it a bit faster. Finally, M1 is the slowest, as it involves a join between the `persons` and `companies` collections, followed by an `unwind` and projection, introducing additional overhead.

Q2 *M3>M2>M1*

Among the three models, M3 is the most efficient as it embeds the `staff` array directly within each company document, allowing the query to compute staff size, which is fast and native to MongoDB's document model, and project the company name without joins or lookups. M2 follows in performance, referencing the `company.name` in each person document and using a group-by to sum employees, but it scans more documents than M3. M1 is the least efficient, it first groups by `companyId` summing the number of documents, made a costly lookup for the `company.name` in the companies collection, unwinds and then projects, introducing significant overhead due to joins and array manipulation.

Q3 *M3>M1≈M2*

Despite updating embedded arrays, M3 is fastest due to MongoDB's efficient handling of array operations with array filters. M1 and M2 have comparable performance, as both update the same number of person documents. However, M1 modifies only references to `companies`, whereas M2 must rewrite embedded company data in each person document—adding overhead and increasing the risk of inconsistencies.

Q4 *M1>M3>M2*

M1 is the most efficient for this update, since the `company.name` is stored centrally in the `companies` collection. Updating involves modifying a single field per company—simple and lightweight. M3 stores the same field, but since it also embeds the entire `staff` array, each update rewrites larger documents, increasing memory and I/O usage. M2 is significantly less efficient, as the `company.name` is embedded in all five million person documents. Each must be updated individually, making this the costliest approach.

Conclusions

In MongoDB, the decision between normalization and denormalization depends largely on data access patterns, update frequency, scalability needs, and schema flexibility.

Model M1, which uses normalized collections with references, is ideal for systems where shared data (e.g., company info) is updated frequently, as it allows centralized updates, avoids redundancy, and scales well in write-heavy environments. It also offers better support for schema evolution, since changes can be made in one place without modifying multiple documents. However, M1 may suffer from slower reads due to the need for joins via aggregation.

On the other hand, Models M2 and M3 adopt denormalization by embedding related data directly—M2 embeds companies in person documents, while M3 embeds persons in companies. These structures improve read performance and reduce query complexity, especially for person- or company-centric queries, but at the cost of update efficiency and scalability. Any change to shared data must be propagated across multiple documents, increasing maintenance overhead and the risk of inconsistencies. Moreover, M3 can face scalability issues when embedding large arrays (e.g., many persons in a company), potentially hitting MongoDB's document size limit.

MongoDB is inherently schema-less, allowing both normalized and denormalized models to evolve. However, the impact of schema changes differs: normalized models make schema updates simpler and safer, while denormalized ones can make them costly and error-prone due to duplicated data. In summary, M1 (normalized) is best suited for update-heavy and evolving systems, offering better scalability and schema flexibility, while M2/M3 (denormalized) are optimized for read-heavy workloads with stable structures.