BSE Barcelona School of Economics

Big Data Management Lab 2

Denis Shadrin, Aleksandr Smolin

May 26, 2025

Table 1: Query Execution Times per Model (seconds)

Model	Q1	Q2	Q3	Q4
M1 (References)	87.83	15.79	47.01	9.50
M2 (Person embeds Company)	12.68	6.30	50.93	76.44
M3 (Company embeds Persons)	6.04	1.25	6.54	17.09

Setup

The benchmark uses a dataset of 500,000 companies and 5,000,000 persons. All data is loaded into a local MongoDB instance (`mongodb://localhost:27017`) in database `lab1_db`. Key indexes created to accelerate queries:

- **companies_ref:** indexes on `name`, `founded`, and a text index on `name` for pattern searches.
- **persons_ref:** indexes on `company_id` (foreign key), `birth_date`, `age`, `last_name`, and a compound index on (`first_name`, `last_name`).
- **persons_emb:** indexes on embedded fields `company._id`, `birth_date`, `age`, compound (`first_name`, `last_name`), and `company.name`.
- **companies_emb:** indexes on `name`, `founded`, embedded `employees.last_name`, `employees.birth_date`, and a text index on `name`.

Answers to Questions

1. **Order for Q1 (best to worst):** M3 < M2 < M1.

   Explanation: Q1 concatenates names and looks up company per person; in M3 employee-company pairs are stored together, so no lookup is needed. M2 still requires nested project on embedded docs, and M1 incurs the most overhead with a lookup per document.

2. **Order for Q2 (best to worst):** M3 < M2 < M1.

   Explanation: M3 is fastest because it reads only 500 K parent documents and computes array sizes in-memory. M2 is next, benefitting from pre-embedded company IDs and a single grouping pass. M1 is slowest due to thousands of individual `$lookup` operations across the large persons collection.

3. **Order for Q3 (worst to best):** M2 > M1 > M3.

   Explanation: Q3 updates all ages for birth dates before 1988. M2 must update each embedded document individually across the persons_emb collection, making it slowest. M1 updates are faster since they update a single field in persons_ref. M3 uses an array filter to update only matching subdocuments within larger documents, resulting in the fewest writes and fastest execution.

4. **Order for Q4 (worst to best):** M2 > M3 > M1.

   Explanation: Q4 appends `Company` to names. In M2 this update must traverse and rewrite every embedded `company.name` for all person docs (5M writes), making it the slowest. M3 requires updating each top-level company document once (500K writes), faster but still more work than M1, which updates only 500K company docs without touching nested arrays.

5. **Conclusions:**

   - Embedding (denormalization) yields significant read/query performance gains when relationships are read frequently (as seen in Q1 and Q2).
   - References (normalization) can simplify updates on a large number of small documents (M1's Q4 is fastest) and avoids document growth issues.
   - Updates that affect nested data (Q3, Q4) perform best when the data is located in a single document (M3) or when the operation scope is minimized.
   - Thus, denormalization favors read-heavy workloads, while normalization may be advantageous for write-heavy or update-heavy workloads.