Barcelona School of Economics

# Assignment 2

**Big Data Management - DSDM - L2-T05**

Maria Simakova (172708)

Moritz Peist (254017)

May 18, 2025

## C Results and Discussion

The following section discusses the query results for each model. Table 1 showcases our performance run results:[1]

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

| | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| M1 | 114.7579 | 5.5505 | 19.0553 | 1.0061 |
| M2 | 9.2571 | 2.6517 | 20.9813 | 50.6679 |
| M3 | 6.2107 | 0.6222 | 9.4313 | 8.7810 |

### 1. Query 1 Performance

**Order queries from best to worst for Q1:** M3 $>$ M2 $>$ M1

Model 3 (M3) performs best because it centralizes all employee data within each company document. For Q1, which retrieves each person's full and company names, M3 efficiently accesses this data with a single `$unwind` operation on the `staff` array. This approach avoids the costly `$lookup` operation required by M1 or the duplication of company data across many documents in M2. MongoDB's internal optimizations for array operations like `$unwind` are highly efficient when working with embedded document arrays.

### 2. Query 2 Performance

**Order queries from best to worst for Q2:** M3 $>$ M2 $>$ M1

Model 3 (M3) excels in this query again because counting employees is a simple `$size` operator applied to the `staff` array–an O(1) operation requiring minimal computation. The M2 approach requires aggregation with `$group` and `$sum` operations, which are more resource-intensive. M1 performs worst as it needs both grouping operations and a costly `$lookup` to join with company information.

### 3. Query 3 Performance

**Order queries from worst to best for Q3:** M2 $\leq$ M1 $<$ M3

Model 3 (M3) outperforms again. While updating embedded documents within arrays requires array filters, MongoDB's optimization for array operations makes this approach more efficient than updating many separate documents. Models M1 and M2 perform similarly since both require updating the same number of individual person-documents with identical operations. The main difference is that M1 updates standalone person documents while M2 updates person documents with embedded company data.

[1]All performance results presented depend entirely on our specific query implementations. Different query strategies or optimizations might yield different relative performance across models. These findings should be interpreted as specific to our implementation rather than as universal MongoDB performance principles.

## 4. Query 4 Performance

**Order queries from worst to best for Q4:** M2 < M3 < M1

Model 1 (M1) performs best by updating only a small number of compact company documents. M3 is moderately efficient, updating the same number of documents as M1 but with a performance penalty because of their larger size, due to embedded employee arrays. M2 performs the worst, requiring updates across thousands of person documents containing embedded company information.

## 5. Normalization vs. Denormalization in MongoDB

The performance results across query types reveal that MongoDB generally favors denormalized data models, but with important nuances:

1. **Data Locality Principle:** For read operations (Q1, Q2), MongoDB performs best when related data is physically stored together, eliminating the need for joins or lookups. This explains why embedded models (M2, M3) outperform the normalized model (M1) for these queries.
2. **Embedding Direction Matters:** The embedding direction significantly impacts performance. Embedding many small entities (people) within fewer larger entities (companies) in M3 is more efficient than the reverse approach in M2 for most operations. This pattern aligns with the natural one-to-many relationship between companies and employees.
3. **Update Performance Trade-offs:** While M3 performed well for updates to person data (Q3), M1 excelled when updating company data (Q4). This demonstrates that normalized models can outperform denormalized ones when updates affect a small subset of documents and avoiding large document overhead is beneficial.
4. **Document Count vs. Size Impact:** Having fewer documents (M1, M3 for companies) outperforms having many documents (M2) for updates, but document size also matters—M1's smaller company documents updated faster than M3's larger ones with embedded arrays in Q4.

For update operations specifically, the best performance comes from:

- Minimizing the total number of documents that need updating
- Considering document size and update frequency when deciding on the embedding strategy
- Balancing between data locality benefits and update overhead costs
- Designing the schema to match the most common query and update patterns

Thus, we can conclude that MongoDB's document-oriented architecture generally favors denormalized data models for read operations, but the optimal model depends on specific access patterns. The M3 model performs best for read operations and person-focused updates, while M1 excels at company-level updates. This demonstrates that, unlike the simplistic "always denormalize" advice, MongoDB requires thoughtful schema design that considers both read and write patterns to achieve optimal performance.