Project 2 - Pokémon world in MongoDB

Group 8: Simone Cotardo, Srimal Fonseka, Arthur Morgan, Stipe Peran

November 12, 2025

1 Project Scenario

Following the first project implemented in Neo4j, we decided to model the Pokémon world using a document-oriented approach in MongoDB. Instead of focusing on graph relationships such as `EVOLVES_TO` or `OWNS`, this second project focuses on storing and querying semistructured documents efficiently.

MongoDB's document model suits this domain well, as Pokémon, trainers, gyms and battles naturally contain nested attributes, arrays and flexible structures that do not require strong normalization.

Our goal was to reconstruct the same universe used in Project 1, but mapped into document collections. The real data used in the project comes from five JSON files:

- `pokemon.json`
- `trainer.json`
- `gym.json`
- `type.json`
- `battles.json`

2 Methodology

2.1 Dataset and Preprocessing

We reused the dataset previously collected for the Neo4j project. The main source was a JSON dataset containing Pokémon information (ID, name, stats, type, and evolution line). Data were cleaned and normalized using Python (Pandas), and exported as multiple collections in MongoDB.

The database was deployed using **MongoDB Compass** for visualization and the **mongo shell** for queries.

2.2 Collections and Schema Design

The document model was designed according to the *Schema-on-Read* philosophy introduced during the NoSQL lectures. Each document stores semistructured data in JSON format, allowing variable fields and embedded subdocuments.

Collections

- **pokemon**: canonical Pokémon entries with attributes, types, and evolutions.
- **trainers**: documents describing each trainer and their owned Pokémon.
- **gyms**: gym entities and their corresponding specialty types.
- **types**: static collection containing the 18 Pokémon types.
- **battles**: match history between trainers and Pokémon.

```
{
"_id": 1,
"name": "Bulbasaur",
"form": "No form",
"type": ["Grass", "Poison"],
"stats": {
    "hp": 45, "attack": 49, "defense": 49,
    "sp_atk": 65, "sp_def": 65, "speed": 45
},
"total": 318,
"description": "Bulbasaur is a Grass/Poison type Pokemon...",
"evolution_line": ["Bulbasaur", "Ivysaur", "Venusaur"]
}
```

This schema reflects MongoDB's flexible document model, supporting nested structures and arrays, unlike the normalized tables of relational databases or edge-based graph storage.

3 Results

3.1 Queries

Twelve representative queries were implemented to demonstrate real-world data retrieval, filtering, and aggregation.

```
db.pokemon.find({ type: "Grass" })
```

```
db.pokemon.find({ total: { $gt: 500 } }, { name: 1, total: 1 })
```

```
db.pokemon.find({ evolution_line: "Venusaur" })
```

```
db.pokemon.aggregate([
{ $unwind: "$type" },
{ $group: { _id: "$type", count: { $sum: 1 } } },
{ $sort: { count: -1 } }
])
```

```
db.pokemon.find().sort({ "stats.speed": -1 }).limit(1)
```

```
db.pokemon.find({}, { name: 1, evolution_line: 1 })
```

```
db.pokemon.find({ type: { $all: ["Grass", "Poison"] } })
```

```
db.pokemon.aggregate([
{ $unwind: "$type" },
{ $group: { _id: "$type", avg_attack: { $avg: "$stats.attack" } }
   },
{ $sort: { avg_attack: -1 } }
])
```

```
db.pokemon.insertOne({
"id": 999,
"name": "Newmon",
"type": ["Fairy"],
"stats": { "hp": 80, "attack": 70, "defense": 60 },
"total": 410
})
```

```
db.pokemon.updateOne(
{ name: "Venusaur" },
{ $set: { description: "Updated␣description␣for␣Venusaur." } }
)
```

```
db.pokemon.deleteMany({ "stats.hp": { $exists: false } })
```

```
db.pokemon.find({}, { name: 1, total: 1 })
.sort({ total: -1 }).limit(3)
```

3.2 Editing Commands

Five additional editing commands were included, demonstrating MongoDB's flexibility for inserts, updates, and deletes. These operations mimic real-world changes such as new Pokémon discovery or stat correction.

4 Discussion

Compared to Neo4j, MongoDB excels in:

- **Data ingestion speed:** loading JSON documents was faster than importing CSVs and building relationships in Neo4j.
- **Flexibility:** the schema-less design allows embedding and easy updates.
- **Horizontal scalability:** MongoDB scales out with sharding, as introduced in the lecture on data architectures [**?**].

However, Neo4j remains more efficient for relationship-heavy queries such as multi-hop evolutions, due to its index-free adjacency. MongoDB required more explicit aggregations for such traversals.

5 Conclusion

This project successfully migrated the Pokémon world model from a graph to a document-oriented representation. We demonstrated how MongoDB can manage complex entities like Pokémon, trainers, and battles within flexible JSON structures. While Neo4j is superior for analyzing relationships, MongoDB provides better performance and simplicity for hierarchical, nested, and document-centric data.

Future work may include integrating the dataset into a web interface or building an analytics dashboard using MongoDB Atlas and Aggregation Pipelines.