Evaluating NoSQL Models on Nuclear Energy Dataset

MongoDB vs Neo4j

Beatrice Mazzocchi, Asia Montico

Ingegneria Informatica, Automatica e Gestionale “Antonio Ruberti”

Data Science

Sapienza University of Rome

30 May 2025

Introduction

This report investigates which NoSQL model is best suited to analyze a global dataset of power plants[1]with a focus on nuclear energy. Our goal is to support complex analytical queries involving numerical aggregations, filtering, spatial analysis, and time-series evaluations.

To achieve this, we evaluated two leading NoSQL paradigms: the document-based **MongoDB** and the graph-based **Neo4j**. Our methodology followed three phases:

1. Assessing **Neo4j**'s potential in handling queries with explicit relationships.
2. Identifying limitations of Neo4j with numeric and temporal analyses.
3. Designing and implementing a more suitable schema in **MongoDB**, optimized for aggregation-heavy queries.

1 Project Scope

The dataset includes power plant data such as fuel type, capacity, coordinates, generation statistics, and national energy outputs. Our research questions span a range of query types:

- Aggregations by fuel type or country
- Temporal trends in generation
- Comparisons across fuel types (e.g., nuclear vs solar)
- Mortality estimates per energy source
- Spatial distribution of nuclear plants

2 Neo4j

Neo4j is a native graph database that stores data in nodes (entities) and relationships (edges) rather than in tables. It is especially powerful for modeling and querying highly connected data,

Queries Not Suitable for Neo4j: Detailed Analysis

Although Neo4j is highly effective for relational structures, most of the queries in this project do not benefit from a graph-based data model. The table below presents the specific queries that are not well suited for Neo4j, along with a technical rationale for each.

[1] `https://www.kaggle.com/datasets/alistairking/nuclear-energy-datasets`

Query #	Title	Technical reason why Neo4j is unsuitable
1	Capacity by fuel type	Requires a **group by** and sum on an attribute; there is no relational structure between the involved nodes. .
2	Top 10 countries by nuclear capacity	Requires aggregation and sorting by capacity; it's a simple tabular analysis, not relational.
3	Percentage of nuclear capacity over total	Requires two aggregations and a comparison of global values
8	Relative error estimated/actual	A simple calculation between two node attributes (`generation_gwh_2017` and `estimated_generation_gwh_2017`), with no relational interaction.

Table 1: Numerical/analytical queries not suitable for the graph model (Neo4j)

Testing Neo4j's Strengths: Query 5

Although most queries do not benefit from Neo4j's model, we identified one that theoretically aligns with its strengths: **Query 5**, which involves analyzing the geographic distribution of nuclear power plants.

This query leverages the explicit relationship between `PowerPlant` and `Country` nodes. The objective was to group nuclear plants into latitude-longitude bands, aggregated by country. This should, in principle, highlight the power of relationship traversal in a graph database.

The following table shows the actual result obtained by running Query 5, presenting the number of plants and their total capacity grouped by geographic bands:

`lat_band`	`lon_band`	`plant_count`	`capacity_mw`	`countries_in_band`
45–50°	0–5°	9	27890.0	{France}
35–40°	125–130°	6	23076.0	{South Korea}
35–40°	135–140°	6	16386.0	{Japan}

MongoDB vs Neo4j in Query 5

Query 5 involves grouping nuclear power plants into *geographic bands*, i.e., latitude and longitude intervals (e.g., 40°–45°). In this context, significant differences emerge between Neo4j's graph model and MongoDB's document model.

The following figure conceptually illustrates the structure of the data used in Neo4j for this query, emphasizing both the spatial dimension (coordinates) and the relationships between nuclear plants and their respective countries:

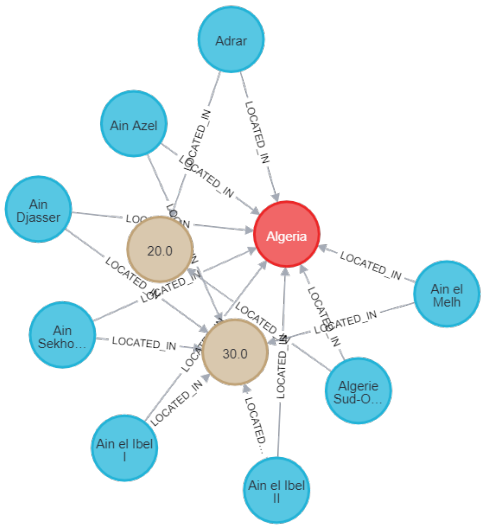


Figure 1: Graph-based representation of nuclear plants grouped by geographic bands (Neo4j)

To better understand the structural and conceptual differences between Neo4j and MongoDB in the context of Query 5, we can break down key aspects of the implementation:

- **Coordinate transformations**: Neo4j is more readable and concise, using `floor()` and string concatenation. MongoDB is more technical and nested, relying on `$floor`, `$concat`, `$toString`, and `$add`.
- **Aggregation**: Neo4j uses `WITH` and `RETURN` for aggregation. MongoDB requires a more complex pipeline using `$group`, `$project`, and `$sort`.
- **Country set handling**: Neo4j handles country collections via `collect(DISTINCT ...)`. In MongoDB, the same logic requires combining `$addToSet` and `$reduce` for sorting and concatenation.
- **Final sorting**: Implemented via `ORDER BY` in Neo4j and via `$sort` in MongoDB.
- **Conceptual effort**: Neo4j is simpler when relationships (e.g., plant–country) are already modeled. MongoDB, while more verbose, remains highly structured and flexible.

In summary:

- **MongoDB** is the most suitable solution for the majority of queries in this project, especially those involving numerical processing, aggregations, and filtering.
- **Neo4j** was successfully used for geospatial band analysis (Query 5), leveraging its relational graph structure where it adds real value.

3 Data Modeling Strategy: Why MongoDB and How We Designed Our Collections

Through our practical analysis (see previous sections), we chose to use **MongoDB** and rather than performing a one-to-one migration of the original six relational tables into six separate MongoDB collections, we designed a new schema that groups together the data that is frequently queried together, improving both performance and semantic cohesion.

Our goal was to create semantically meaningful collections, merging related data sources and reducing fragmentation, so that MongoDB's capabilities—such as nested documents and fast aggregation pipelines—could be fully leveraged. Here is how we structured our collections:

- `death_rates` (from the SQL table `rates_death_from_energy_production_per_twh`): This was intentionally kept as a separate collection because it represents a static and concise dataset—deaths per TWh for each energy source. Since this data rarely changes and includes only a handful of rows, embedding it into other collections would have introduced unnecessary redundancy without any real benefit.

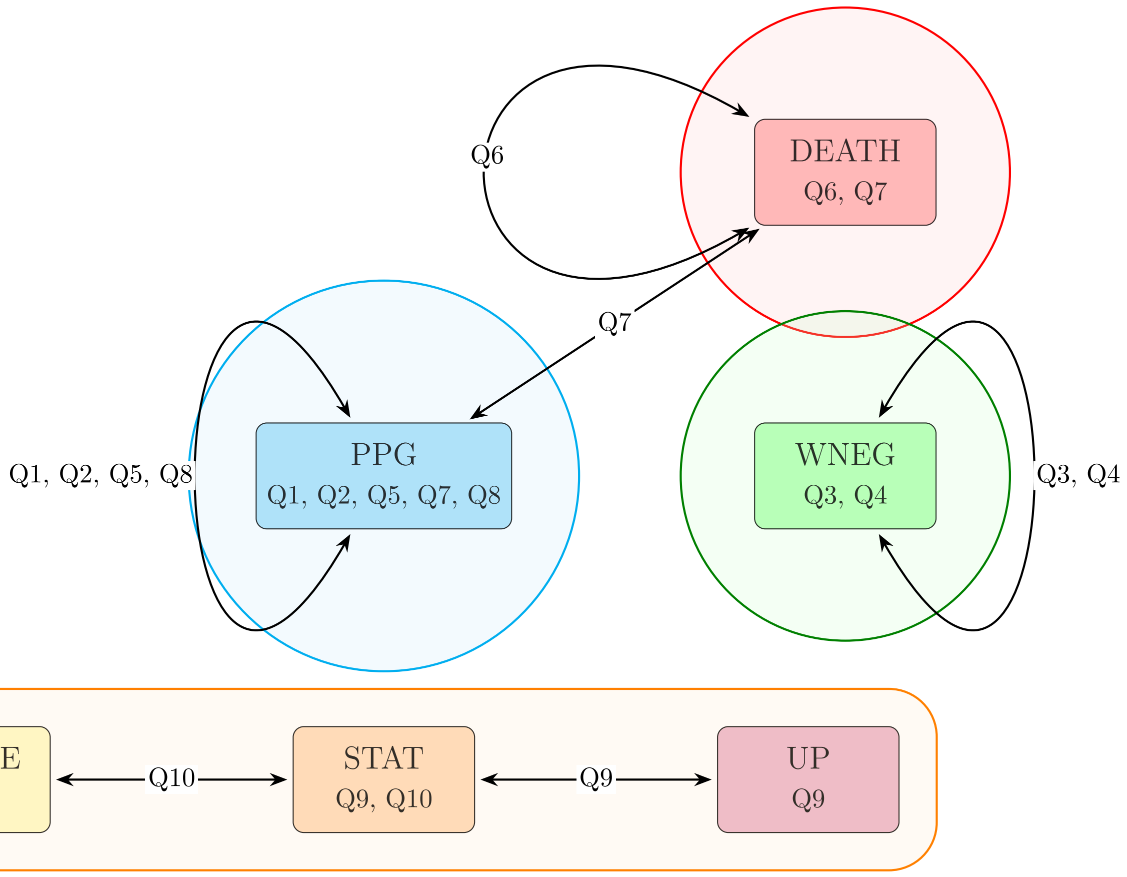
- `power_plant_global`: This collection includes the full dataset of global power plants, including nuclear, coal, gas, solar, and others. It retains key information such as country, geographic coordinates, primary fuel type, and estimated/actual energy production for 2017. This collection supports spatial analysis (as seen in Query 5), fuel-based aggregation (Query 1), and comparisons of nuclear versus solar production (Query 7), among others.

- `world_nuclear_generation`: This dataset is logically distinct from `power_plant_global`, as it represents national-level statistics rather than individual plants. It includes annual figures for each country, such as total TWh generated from nuclear and its share of the country's electricity mix. This data supports temporal and cross-national analyses like those in Query 3 (nuclear energy share) and Query 4 (year-over-year nuclear production increases).

- `us_nuclear_stats`: Instead of keeping three separate collections for U.S.-specific data (`uranium_production_summary_us`, `uranium_purchase_price_us`, and `us_nuclear_generat` we merged them into a unified structure. These datasets share a common key—the year—and together offer a holistic view of the U.S. nuclear sector: from uranium mining and purchases to generation capacity and output. Combining them simplifies queries such as Query 9 (uranium production vs nuclear output) and Query 10 (uranium cost vs generation).

From SQL Tables to MongoDB Collections

Below, we present our selected collection design using a graph model, illustrating how datasets and queries are interrelated across domain



PPG	Power Plant Global
WNEG	World Nuclear Energy Gen.
DEATH	Death Rates per Energy Source
STAT	US Nuclear Statistics
UPRICE	Uranium Purchase Price
UP	Uranium Production

Figure 2: Graph of table interactions where each edge represents co-occurrence in queries. Self-loops represent table self-interactions. All queries (Q1–Q10) are included.

3.1 Data Loading into MongoDB

To load the datasets into MongoDB, a dedicated Python script was implemented using the `pandas` and `pymongo` libraries. Each dataset was transformed into a MongoDB-compatible format, and the collections were populated in a controlled manner.

The most complex collection is `us_nuclear_stats`, which integrates data from three different sources:

- **US Nuclear Generating Statistics** (1971–2021): annual data on nuclear electricity generation.
- **Uranium Production Summary** (available from 2009 onward).
- **Uranium Purchase Price** (available from 2002 onward).

Since only the first dataset covers the entire period (1971–2021), a *nested* data structure was designed where each document corresponds to a year and contains:

- The nuclear generation data (`nuclear_generation`), which is always present.
- The uranium production and price data (`uranium_production` and `uranium_prices`), included only if available for that specific year.

This design choice ensures:

- **Temporal continuity** and data coherence across the full time span.
- Prevention of **information loss** that would occur if only overlapping years among datasets were kept.

Example Document Stored in `us_nuclear_stats`

```
{
  "_id": {
    "$oid": "683330e8744d3bc5ba3c670e"
  },
  "year": 2009,
  "uranium_production": {
    "mine_production_lbs_mill": "4.1",
    "concentrate_production_lbs_mill": "3.7",
    "concentrate_shipments_lbs_mill": "3.6",
    "employment_person_years": "1,096"
  },
  "uranium_prices": {
    "total_purchased_usd_per_lb": 45.86,
    "us_producers_usd_per_lb": 0,
    "us_brokers_usd_per_lb": 41.88,
    "other_us_suppliers_usd_per_lb": 0,
    "foreign_suppliers_usd_per_lb": 46.68,
    "us_origin_usd_per_lb": 48.92,
    "foreign_origin_usd_per_lb": 45.35,
    "spot_contracts_usd_per_lb": 46.45,
    "term_contracts_usd_per_lb": 45.74
  },
  "nuclear_generation": {
    "total_generation_mwh": 3950330926,
    "nuclear_generation_mwh": 798854585,
    "nuclear_fuel_share_pct": 20.2,
    "capacity_factor_pct": 90.3,
    "summer_capacity_mw": 101004
  }
}
```