Pokemon Database Report

Dhvani Thakkar

May 2024

1 Introduction

The objective of this project was to explore GenAI's capabilities in database design using two different paradigms: SQL and NoSQL. The chosen problem domain was a simplified version of the Pokemon game, focusing on core mechanics like Pokemon types, moves, and type effectiveness. The task involved creating a database schema and populating it with given data. The challenge was to handle the many-to-many relationship between Pokemon and Moves. The task was performed twice, once using SQL and once using NoSQL (MongoDB), to compare the results.

2 Requirements

In Pokemon, each creature can have one or two 'types'. These types determine their effectiveness against other Pokemon types. Every Pokemon has at least a primary type, and some have a secondary type. The game-play involves using moves to attack other Pokemon. Each move has a specific power and type. Every move can be learned by a set of Pokemon, and each Pokemon has a set of moves they can learn. At a minimum, we need to create database tables (or equivalent structures in non-relational databases) to store Poke- mon, Type, and Move. However, 'Pokemon' and 'Move' have a classic many-to-many relationship, which needs to be handled appropriately. The tasks are as follows:

1. Create all the necessary tables (or equivalent structures).
2. Populate the tables with the following details:
    - Details about various Pokemon and their types.
    - Details about various moves, their power, and type.
    - Information about which Pokemon can learn which moves.
    - Information about the effectiveness of different types against each other.
3. Write a query to return all the Pokemon who can learn the move 'Return'.
4. Write a query to return all the moves in the game that are powerful against the Grass type.

3 MySQL: Relational SQL-oriented Approach

I used MySQL Workbench for the first time, having only used the Command line before. I found it helpful as I could instantly see the EER diagram and edit + save my queries easily.

I started by giving GenAI the problem statement, but soon modified it to explain the SQL structure to GenAI as I understood it:

> "Give me a MongoDB database design for Pokemon, with the functionality: a pokemon has unique name and gets 1 primary and optional 1 secondary type, and can learn n skills. Each skill has given strength and type. All types are powerful or weak against one another, and if there is a skill of a type more powerful than another skill of less powerful type, the more powerful skill strength doubles for that move, and the less powerful skill strength becomes half for that move."

ChatGPT returned some extra, unnecessary tables for the database.

Gemini gave tables that were a very good baseline for creating the MySQL database. I asked to first give a table design, then data given to populate, and once satisfied with those, the two queries.

3.0.1 Table Definition:

```
CREATE TABLE Typ (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) UNIQUE
);

CREATE TABLE Pokemon (
  id INT AUTO_INCREMENT,
  name VARCHAR(255) UNIQUE,
  primary_type_id INT NOT NULL,
  secondary_type_id INT DEFAULT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (primary_type_id) REFERENCES Typ(id),
  FOREIGN KEY (secondary_type_id) REFERENCES Typ(id)
);

CREATE TABLE Skill (
  id INT AUTO_INCREMENT,
  name VARCHAR(255) UNIQUE,
  strength INT,
  type_id INT NOT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (type_id) REFERENCES Typ(id)
);

CREATE TABLE Pokemon_Skill (
  pokemon_id INT NOT NULL,
  skill_id INT NOT NULL,
  is_boosted BOOLEAN DEFAULT FALSE,
  FOREIGN KEY (pokemon_id) REFERENCES Pokemon(id),
  FOREIGN KEY (skill_id) REFERENCES Skill(id),
  PRIMARY KEY (pokemon_id, skill_id)
);

CREATE TABLE type_matchup (
  attacking_type_id INT NOT NULL,
  defending_type_id INT NOT NULL,
  multiplier DECIMAL(2,1) NOT NULL,
  FOREIGN KEY (attacking_type_id) REFERENCES typ(id),
  FOREIGN KEY (defending_type_id) REFERENCES typ(id),
  PRIMARY KEY (attacking_type_id, defending_type_id)
);
```

3.0.2 Queries:

```
SELECT p.name AS pokemon
FROM Pokemon p
INNER JOIN Pokemon_Skill ps ON ps.pokemon_id = p.id
INNER JOIN Skill s ON ps.skill_id = s.id
WHERE s.name = "Return";
```

```
SELECT s.id, s.name
FROM skill s
WHERE type_id IN
(SELECT attacking_type_id
FROM type_matchup
WHERE defending_type_id = (SELECT id FROM typ WHERE name = "Grass")
AND multiplier = 2);
```

4 MongoDB: Document NoSQL-oriented Approach

I used both ChatGPT and Gemini to find MongoDB queries, given that I had no prior experience with NoSQL databases.

4.1 Gemini

I was able to understand how documents are structured, where and how queries of different formats (such as aggregation pipeline, simple find queries and node.js terminal queries).

Whether querying with a more SQL-leaning explanation or the problem statement itself, Gemini was very precise and Relational in its database creation.

- Defined primary_type and secondary_type separately for the Pokemon table, instead of an array of types.
- Used ObjectIDs extensively as foreign keys to reference other collections.
- Tried to define its own objectIDs, which MongoDB does not allow, so it took a long time to populate the collections with the objectIDs generated by MongoDB in the foreign collections.

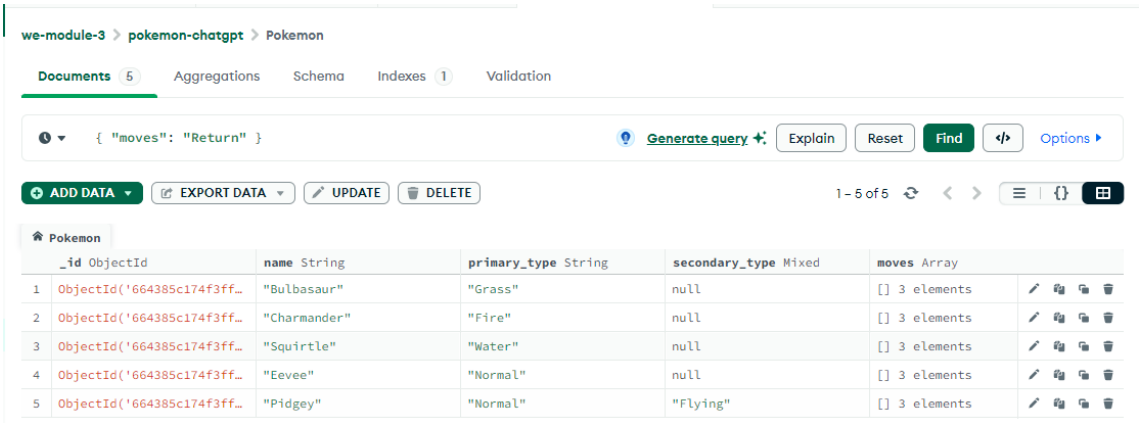
This proved to be unnecessary, time consuming and very similar to the SQL approach, so I moved to with ChatGPT.

4.2 ChatGPT

When I used the query modified to better explain the SQL database requirements for MongoDB, ChatGPT used ObjectIDs as foreign keys too, but when given just the problem statement, it was able to provide very good NoSQL collection definitions and queries.

4.2.1 Query 1:

```
{ "moves": "Return" }
```



4.2.2 Query 2:

I used the aggregation pipeline in MongoDB Compass to run the second query, and learned to use lookup, unwind, match and project stages, as well as how they translate to similar SQL queries.

```
[
  {
    $lookup: {
      from: "Type",
      localField: "type",
      foreignField: "name",
      as: "move_type"
    }
  },
  {
    $unwind: "$move_type"
  },

  {
    $match: {
      "move_type.strengths": "Grass"
    }
  },
  {
    $project: {
      _id: 0,
      name: 1,
      type: 1
    }
  }
]
```

```
name : "Ember"
type : "Fire"
```

```
name : "Wing Attack"
type : "Flying"
```

5 Differences between SQL & NoSQL

5.0.1 Schema Design:

- MySQL: Required a predefined schema with tables, columns, and relationships defined in advance. Tables needed to be normalized to avoid redundancy and maintain data integrity, such as Type, Skill name definitions.
- MongoDB: flexible schema design. Documents can have varying structures, and fields can be added dynamically, such as for Pokemon Type. There's no need for predefined schema or normalization.

5.0.2 Data Model:

- MySQL: Follows a relational data model with tables linked by foreign keys to establish relationships. All tables are interlinked.
- MongoDB: Follows a document-oriented data model where data is stored in JSON-like documents within collections. Relationships were called upon through matching and references.

5.0.3 Scalability:

- MySQL: Typically scaled vertically by increasing server resources (CPU, RAM, storage).
- MongoDB: Scales horizontally, distributing data across multiple servers or clusters to handle large volumes of data and high throughput.

5.0.4 Performance:

- MySQL: Provides high performance for structured data with complex relationships and joins.
- MongoDB: Offers high performance for unstructured or semi-structured data with flexible schema and simple queries.

6 Conclusions

- GenAI was quick to understand the tables needed for MySQL, given there was adequate explanation of SQL requirements.
- The MongoDB pipeline query confused some collection names and attributes of documents it created itself.
- Gemini could store the ObjectIDs given to it and apply them to MongoDB queries till 4 subsequent prompts. Once re-prompted the IDs and re-stressed the format of queries needed, Gemini continued with the changes throughout.
- ChatGPT seemed to perform better with MongoDB and Gemini seemed better with MySQL for this task.
- It was very easy to learn usage of new platforms when taking the help of both GenAI and documentation, from installation to helpful tools.