Designing a database for storing Pokemon data in SQL and NoSQL using Copilot

Anvesha Singh

May 9, 2024

Contents

1 Introduction

The goal is to design a Pokemon Database that contains data regarding Pokemon, moves of those pokemons, and the types of the moves and pokemons using **Copilot**. The database will be implemented using **SQL** (table structured data) and **NoSQL** (unstructured data).
This report focuses on the difference in experience when using Copilot to design an SQL database and then a NoSQL database. I will also document the difference between SQL and NoSQL and how it affects the ease of using GenAI.

Details of the database:
Pokemon can have one or two 'types' which decide whether they're more effective or less effective against other Pokemon types. Every pokemon has a primary type; some also have a secondary type.
The game involves using moves to attack other Pokemon, and each move has a certain power and type. Every move has a set of Pokemon who are capable of learning it; and every Pokemon has a set of moves it can learn.
To deal with the many-to-many relationship between Pokemon and Moves, we create a separate storage to store and relate all the different Pokemons and moves.
Tasks:

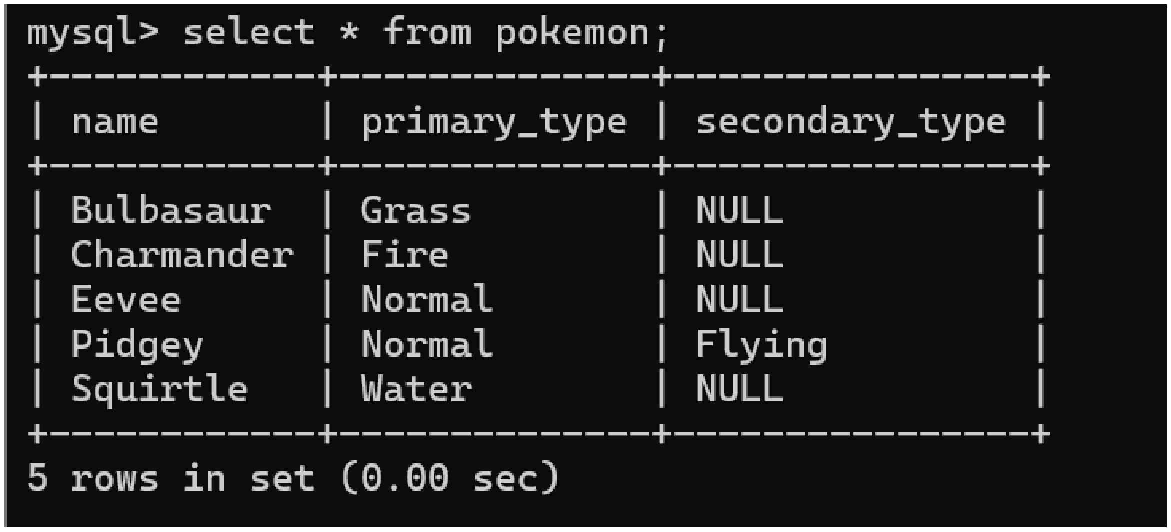
- Create all the tables needed and populate them.
- Write a query that returns all the pokemon who can learn 'Return'.
- Write a query that returns all the moves in the game that are powerful against Grass.

2 SQL

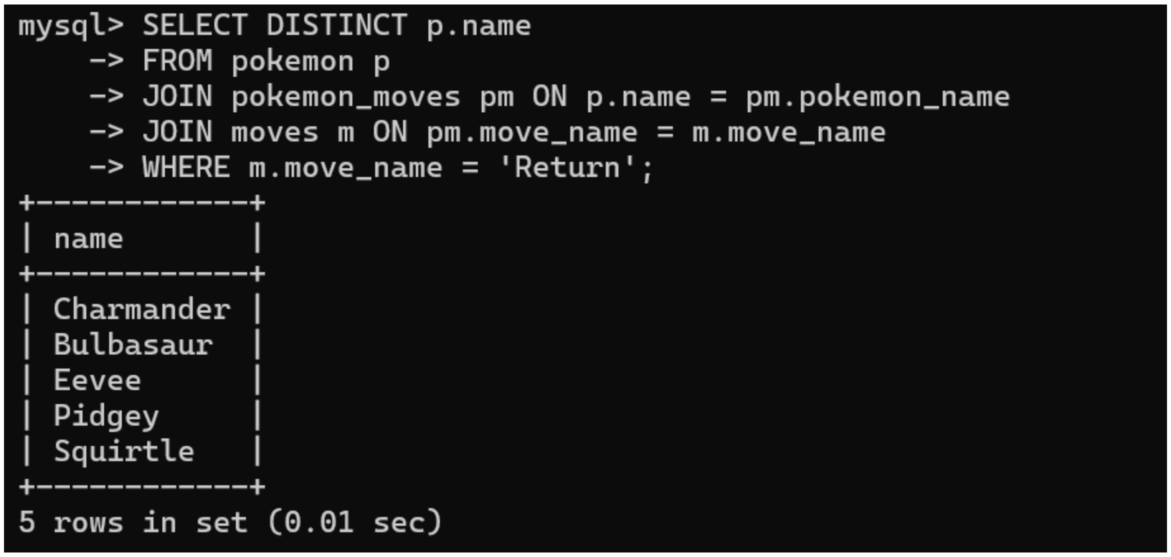
SQL (Structured Query Language) is a domain-specific language used to manage data, especially in relational database management systems (RDBMS). It allows users to store, retrieve, modify, and analyze data efficiently using tables. SQL provides standardized commands for querying and manipulating databases, making it a fundamental tool for developers and data professionals.

Experience with GenAI:

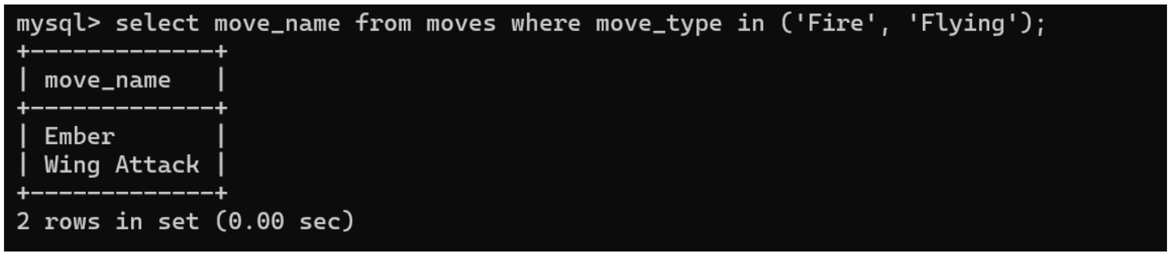
- Copilot was easily able to identify all the tables required although it missed a few details hence I had to keep a keen eye at each step to check whether it has implemented everything or not.
- Compared to ChatGPT and Gemini, the memory of Copilot is way lower as it kept forgetting already implemented changes and it would provide basic completely unrelated examples from the web when asked for commands.
- Interestingly, small keyword changes like 'insert' to 'enter' would change the structure of the query.
- It also made an extreme logical mistake and provided the query that found Moves that were **weak against Grass** instead of being strong.



```
mysql> select * from pokemon;
+------------+--------------+----------------+
| name       | primary_type | secondary_type |
+------------+--------------+----------------+
| Bulbasaur  | Grass        | NULL           |
| Charmander | Fire         | NULL           |
| Eevee      | Normal       | NULL           |
| Pidgey     | Normal       | Flying         |
| Squirtle   | Water        | NULL           |
+------------+--------------+----------------+
5 rows in set (0.00 sec)
```



```
mysql> SELECT DISTINCT p.name
    -> FROM pokemon p
    -> JOIN pokemon_moves pm ON p.name = pm.pokemon_name
    -> JOIN moves m ON pm.move_name = m.move_name
    -> WHERE m.move_name = 'Return';
+------------+
| name       |
+------------+
| Charmander |
| Bulbasaur  |
| Eevee      |
| Pidgey     |
| Squirtle   |
+------------+
5 rows in set (0.01 sec)
```



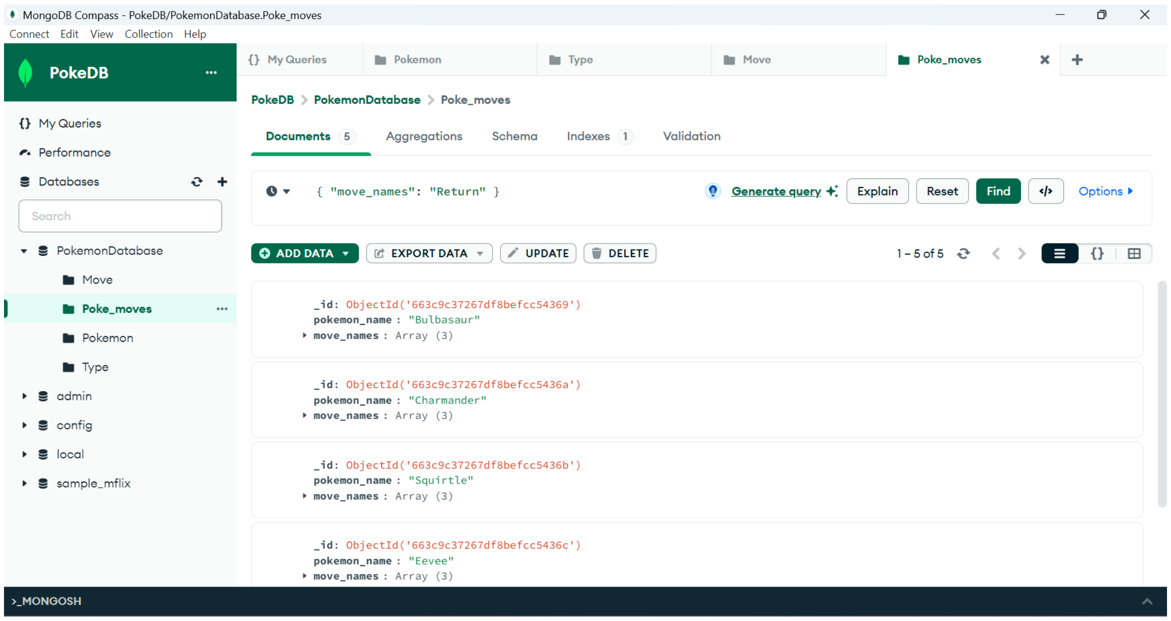
```
mysql> select move_name from moves where move_type in ('Fire', 'Flying');
+-------------+
| move_name   |
+-------------+
| Ember       |
| Wing Attack |
+-------------+
2 rows in set (0.00 sec)
```

3 NoSQL

NoSQL ditches the rigid tables of traditional databases. It stores data in flexible formats like documents or key-value pairs. This makes it perfect for big, messy data that won't fit neatly in rows and columns. Imagine it as a filing system with adaptable containers, instead of pre-defined folders. Great for scalability and handling ever-changing data, but it trades some of the structure and consistency of relational databases.

Experience with GenAI:

- Again, Copilot was easily able to identify all the tables required although it again missed the same details as it did for sql which are the strong-against and weak-against attributes of each type.
- Although the memory felt slightly better this time, it was not able to solve any of my doubts regarding usage of MongoDB Compass.
- It kept giving Mongosh commands even though I asked for the Find tab commands without even mentioning where I'm supposed to use the command. The AI assumes that we're already very much comfortable with the application being used.
- Although it's foundation was good and was able to pretty much finish the assignment in just a few prompts, it suffered from severe hallucination and amnesia.



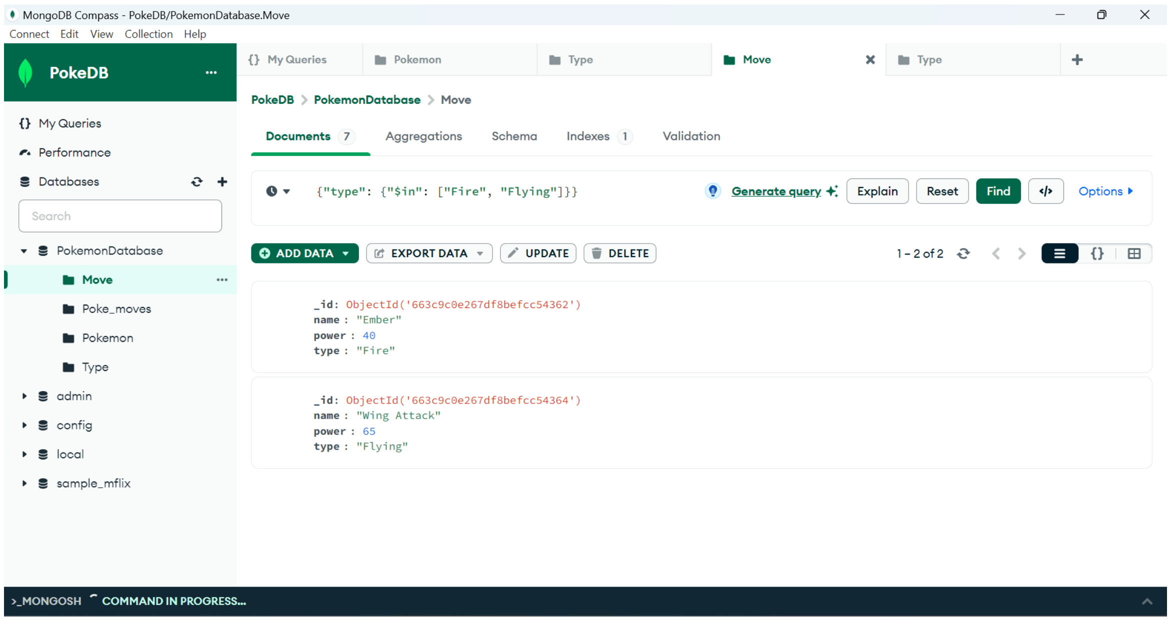
```
{ "move_names": "Return" }
```

```
_id: ObjectId('663c9c37267df8befcc54369')
pokemon_name : "Bulbasaur"
move_names : Array (3)

_id: ObjectId('663c9c37267df8befcc5436a')
pokemon_name : "Charmander"
move_names : Array (3)

_id: ObjectId('663c9c37267df8befcc5436b')
pokemon_name : "Squirtle"
move_names : Array (3)

_id: ObjectId('663c9c37267df8befcc5436c')
pokemon_name : "Eevee"
move_names : Array (3)
```



```
{"type": {"$in": ["Fire", "Flying"]}}
```

```
_id: ObjectId('663c9c0e267df8befcc54362')
name : "Ember"
power : 40
type : "Fire"

_id: ObjectId('663c9c0e267df8befcc54364')
name : "Wing Attack"
power : 65
type : "Flying"
```

4 Conclusion

Overall, the AI's behaviour for both the scenarios was similar for both the SQL and NoSQL databases. Although it was thorough with the syntax and basics of each category, it was extremely bad at guiding the user through the whole procedure and it is recommended to watch a few tutorials on youtube beforehand and only using Copilot for code-writing. However, Copilot provided some useful resources as references with each prompt.

5 References

- Git Repo Link
- SQL Chat Doc
- NoSQL Chat Doc