# Pokemon Database Design

Kashish Bhurani

May 10, 2024

## 1 Introduction

We were assigned the task of Database Design using SQL and NoSQL databases for the game of Pokemon. We were randomly allotted SQL or NoSQL which provided a chance to explore what we did not know. This report is about my experience and what I learnt working with SQL and NoSQL databases with the help of Generative AI.

## 2 Methodology

### 2.1 Approach

I initially had no clue about NoSQL databases and started to explore more about them by learning about a few NoSQL databases like MongoDB and OrientDB. I learnt that MongoDB was the easiest to use and beginner friendly and started by downloading MongoDB Compass.

I first started with working with MySQL since I knew a little about it. After reading the instructions about the Pokemon game, I had a rough idea about the tables and the relations in between them and started with it. I asked ChatGPT to elaborate it for me and it helped me write proper queries for the table. After designing the entire database in MySQL I was satisfied with it after a few changes.

Now that I had to start working with NoSQL the first thing i noticed was that NoSQL databases don't have tables and work very differently than SQL databases. That's when I referred the MongoDB documentation and got along with it.

I asked Gen AI to help me write the queries and it did that with the help of the MySQL code that I had already worked for me. That helped me draw parallels in between the two kinds of databases and that's when I understood quite a lot of things,

## 3 SQL and NoSQL code snippets

### 3.1 SQL

```
CREATE TABLE Pokemon (
    PokemonID INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    PrimaryType VARCHAR(50) NOT NULL,
    SecondaryType VARCHAR(50),
    CONSTRAINT uc_PokemonName UNIQUE (Name)
);

INSERT INTO Pokemon (Name, PrimaryType, SecondaryType) VALUES
('Bulbasaur', 'Grass', NULL),
('Charmander', 'Fire', NULL),
('Squirtle', 'Water', NULL),
```

```
    ('Eevee', 'Normal', NULL),
    ('Pidgey', 'Normal', 'Flying');

    SELECT p.Name AS PokemonName
    FROM Pokemon p
    JOIN PokemonMoves pm ON p.PokemonID = pm.PokemonID
    JOIN Move m ON pm.MoveID = m.MoveID
    WHERE m.Name = 'Return';
```

### 3.2 NoSQL

```
    db.createCollection("Pokemon")

    db.Pokemon.insertMany([
  { "_id": 1, "Name": "Bulbasaur", "PrimaryType": "Grass", "SecondaryType": null },
  { "_id": 2, "Name": "Charmander", "PrimaryType": "Fire", "SecondaryType": null },
  { "_id": 3, "Name": "Squirtle", "PrimaryType": "Water", "SecondaryType": null },
  { "_id": 4, "Name": "Eevee", "PrimaryType": "Normal", "SecondaryType": null },
  { "_id": 5, "Name": "Pidgey", "PrimaryType": "Normal", "SecondaryType": "Flying" }
])

        db.Move.find({
      Type: { $in: ["Fire", "Flying"] }
    }, {
      Name: 1,
      Power: 1,
      Type: 1,
      _id: 0
    })
```

## 4 Things I Learnt

### 4.1 Structure of these Databases

- SQL uses predefined schema in the form of tables while in NoSQL databases data is stored in many ways which means it can be document-oriented, column-oriented, graph-based, or organized as a key-value store.
- This gives flexibility and a dynamic structure to NoSQL databases
- I felt that I could do a lot of nested queries and insertions in NoSQL databases as it was able to form a hierarchy but i found it difficult in SQL.

### 4.2 Use Cases

- Both SQL and NoSQL databases cater to different use cases based on the type of work you have
- I realised that if I have large unstructured data NoSQL would be a better option and if the data is small has proper well defined relations, SQL is easy and more effective.

## 5 Conclusion

The exploration of SQL and NoSQL databases within the context of designing Pokemon database provided valuable insights into their working, respective strengths and applications. Using generative AI tools made the learning process faster and helped me transition between SQL and NoSQL databases. After this assignment I feel I am better equipped to decide which one to use based on the work I have and explore more complex database designs.