# Pokemon Database Report

Lalithya Varma Samanthapudi

May 9, 2024

## 1 Introduction

This report embarks on simplifying Pokémon mechanics through effective database management. By creating a database system to handle Pokémon attributes like types and moves, it aims to streamline gameplay essentials. Delving into the design and population of tables, alongside crafting queries to extract pertinent information, the report endeavors to offer insights into optimizing Pokémon data organization. Ultimately, it seeks to enhance the gaming experience and facilitate efficient data retrieval for Pokémon enthusiasts.

## 2 Problem Statement

The task involves simplifying Pokémon mechanics by modeling Pokémon types, moves, and their relationships. This includes designing a database schema and executing queries to retrieve Pokémon capable of learning specific moves and moves powerful against certain Pokémon types.

## 3 Database Design

### 3.1 Tables

- **Pokemon:** This table stores information about each Pokemon, including its ID, name, primary type, and secondary type (if applicable).
- **Type:** This table contains the different types of Pokemon, such as Grass, Fire, Water, etc.
- **Move:** Moves are the actions that Pokemon can perform during battles. This table stores information about each move, including its ID, name, power, and associated type.
- **PokemonMove:** As Pokemon can learn multiple moves, and moves can be learned by multiple Pokemon, this table establishes a many-to-many relationship between Pokemon and moves.

### 3.2 Creating Tables

#### 3.2.1 SQL

```
CREATE TABLE Pokemon (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    primary_type VARCHAR(20),
    secondary_type VARCHAR(20)
);

CREATE TABLE Type (
    id INT PRIMARY KEY,
    name VARCHAR(20)
);

CREATE TABLE Move (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    power INT,
    type_id INT,
    FOREIGN KEY (type_id) REFERENCES Type(id)
);

CREATE TABLE PokemonMove (
    pokemon_id INT,
    move_id INT,
    FOREIGN KEY (pokemon_id) REFERENCES Pokemon(id),
    FOREIGN KEY (move_id) REFERENCES Move(id),
    PRIMARY KEY (pokemon_id, move_id)
);
```

### 3.2.2 NoSQL

In NoSQL databases like MongoDB, you typically work with collections rather than tables, and queries are expressed using MongoDB's query language. Here's how you would perform the equivalent tasks in a NoSQL context:

```
javascript
-- Create collections for Pokémon, Types, and Moves
db.createCollection("pokemon");
db.createCollection("types");
db.createCollection("moves");
```

## 3.3 Querying Data

To retrieve specific information from the database, SQL queries are used. Below are examples of SQL queries for querying Pokemon data:

### 3.3.1 SQL

```
-- Populating Pokemon table
INSERT INTO Pokemon (id, name, primary_type, secondary_type) VALUES (1, 'Bulbasaur', 'Grass', NULL);
INSERT INTO Pokemon (id, name, primary_type, secondary_type) VALUES (2, 'Charmander', 'Fire', NULL);
INSERT INTO Pokemon (id, name, primary_type, secondary_type) VALUES (3, 'Squirtle', 'Water', NULL);
INSERT INTO Pokemon (id, name, primary_type, secondary_type) VALUES (4, 'Eevee', 'Normal', NULL);
INSERT INTO Pokemon (id, name, primary_type, secondary_type) VALUES (5, 'Pidgey', 'Normal', 'Flying');

-- Populating Move table
INSERT INTO Move (id, name, power, type_id) VALUES (1, 'Tackle', 35, 1);
INSERT INTO Move (id, name, power, type_id) VALUES (2, 'Vine Whip', 40, 4);
INSERT INTO Move (id, name, power, type_id) VALUES (3, 'Return', 100, 1);
INSERT INTO Move (id, name, power, type_id) VALUES (4, 'Ember', 40, 2);
INSERT INTO Move (id, name, power, type_id) VALUES (5, 'Water Gun', 40, 3);
INSERT INTO Move (id, name, power, type_id) VALUES (6, 'Wing Attack', 65, 5);
INSERT INTO Move (id, name, power, type_id) VALUES (7, 'Headbutt', 70, 1);

-- Populating PokemonMove table
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (1, 1);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (1, 2);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (1, 3);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (2, 1);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (2, 4);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (2, 3);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (3, 1);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (3, 5);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (3, 3);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (4, 1);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (4, 7);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (4, 3);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (5, 1);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (5, 6);
INSERT INTO PokemonMove (pokemon_id, move_id) VALUES (5, 3);

-- Populating Type table
INSERT INTO Type (id, name) VALUES (1, 'Normal');
INSERT INTO Type (id, name) VALUES (2, 'Fire');
INSERT INTO Type (id, name) VALUES (3, 'Water');
INSERT INTO Type (id, name) VALUES (4, 'Grass');
INSERT INTO Type (id, name) VALUES (5, 'Flying');
```

#### 3.3.2 NoSQL

```
javascript
--Insert documents into the 'pokemon' collection
db.pokemon.insertMany([
    { name: "Bulbasaur", types: ["Grass"], moves: ["Tackle", "Vine Whip", "Return"] },
    { name: "Charmander", types: ["Fire"], moves: ["Tackle", "Ember", "Return"] },
    { name: "Squirtle", types: ["Water"], moves: ["Tackle", "Water Gun", "Return"] },
    { name: "Eevee", types: ["Normal"], moves: ["Tackle", "Headbutt", "Return"] },
    { name: "Pidgey", types: ["Normal", "Flying"], moves: ["Tackle", "Wing Attack", "Return"] }
]);

--Insert documents into the 'types' collection (if necessary)
db.types.insertMany([
    { name: "Grass" },
    { name: "Fire" },
    { name: "Water" },
    { name: "Normal" },
    { name: "Flying" }
]);

--Insert documents into the 'moves' collection (if necessary)
db.moves.insertMany([
    { name: "Tackle", power: 35, type: "Normal" },
    { name: "Water Gun", power: 40, type: "Water" },
    { name: "Ember", power: 40, type: "Fire" },
    { name: "Vine Whip", power: 40, type: "Grass" },
    { name: "Wing Attack", power: 65, type: "Flying" },
    { name: "Headbutt", power: 70, type: "Normal" },
    { name: "Return", power: 100, type: "Normal" }
]);
```

### 3.4 Querying Data

To retrieve specific information from the database, SQL queries are used. Below are examples of SQL queries for querying Pokemon data:

#### 3.4.1 SQL

```
-- Query to return all Pokemon who can learn 'Return'
SELECT p.name
FROM Pokemon p
JOIN PokemonMove pm ON p.id = pm.pokemon_id
JOIN Move m ON pm.move_id = m.id
WHERE m.name = 'Return';

-- Query to return all moves in the game that are powerful against Grass
SELECT m.name
FROM Move m
```

```
JOIN Type t ON m.type_id = t.id
WHERE t.name IN ('Fire', 'Flying');
```

#### 3.4.2 NoSQL

```
-- Querying for Pokémon Capable of Learning 'Return':
javascript
db.pokemon.find({ "moves": "Return" }, { "_id": 0, "name": 1 })

-- Querying for Moves Powerful Against Grass-type Pokémon:
javascript
db.moves.find({ "type": { $in: ["Fire", "Flying"] }, "name": { $ne: "Return" } }, { "_id": 0, "name": 1 })
```

## 4 SQL vs. NoSQL

### 4.1 Overview

In the realm of database management, two primary systems stand out: SQL (Structured Query Language) and NoSQL (Not Only SQL). Each system possesses distinct attributes and caters to specific needs.

### 4.2 Contrasts

SQL databases adhere to a relational model and employ structured query language (SQL) for data retrieval. Conversely, NoSQL databases operate without a fixed schema and excel in handling unstructured data efficiently.

### 4.3 Advantages and Drawbacks

SQL databases boast robust consistency and adeptness in handling intricate queries, albeit they may exhibit limitations in scalability. NoSQL databases, on the contrary, offer remarkable scalability and adaptability, albeit potentially at the cost of consistency.

### 4.4 Application Scenarios

SQL databases find favor in applications necessitating ACID (Atomicity, Consistency, Isolation, Durability) properties, such as financial systems. NoSQL databases shine in environments grappling with copious amounts of unstructured data, such as social media platforms.

## 5 Conclusion

In summary, creating a well-designed database is essential for efficiently managing intricate data structures like Pokémon attributes. Through meticulous design and implementation of a structured database system, the complexities of storing and accessing Pokémon data can be effectively managed and overcome.