Evaluating GenAI's ability to aid users in creating databases on both MySQL and NoSQL platforms

Gen AI: ChatGPT

Aishvarya Jaiswal, 8.5.2024

1. Abstract

This report evaluates the effectiveness of GenAI in assisting users in creating databases using MySQL and NoSQL (MongoDB). The findings suggest that GenAI offers more effective guidance for MySQL compared to MongoDB. Users reported a smoother experience with MySQL, whereas GenAI appeared less adept and more confused when assisting with MongoDB. These observations underscore the challenges GenAI encounters in adapting to diverse database systems. Recognizing these distinctions can inform efforts to enhance GenAI's support for NoSQL databases such as MongoDB.

2. Problem Statement

To explore GenAI's capability with databases, our task involved creating a database for the popular "Pokemon" video game series. We were provided with simplified game data, including information about Pokemon types, moves, and their relationships. Specifically:

Pokemon can possess one or two types, influencing their effectiveness against other types. Each Pokemon has a primary type, with some having a secondary type. The game revolves around using moves to attack other Pokemon, with each move having a power and type. Every move is associated with a set of Pokemon capable of learning it, and each Pokemon can learn a set of moves.

Question: At the very least, we'd need database tables to store Pokemon, Type, and Move.
However, 'Pokemon' and 'Move' have a classic many-to-many relationship. How can we deal with this?

• Create all the tables needed. (5)
  – Pokemon
  – Type
  – Move

• With the following details, populate the tables: (5)
  – Bulbasaur is a pokemon of Grass type.
  – Charmander is a pokemon of Fire type.
  – Squirtle is a pokemon of Water type.
  – Eevee is a pokemon of Normal type.
  – Pidgey is a pokemon of the Normal/Flying type.
  – Bulbasaur can learn Tackle, Vine Whip, and Return.
  – Charmander can learn Tackle, Ember, and Return.
  – Squirtle can learn Tackle, Water Gun, and Return.
  – Eevee can learn Tackle, Headbutt, and Return.
  – Pidgey can learn Tackle, Wing Attack, and Return.
  – Tackle has 35 power and is Normal type.
  – Water Gun has 40 power and is Water type.
  – Ember has 40 power and is Fire type.
  – Vine Whip has 40 power and is Grass type.
  – Wing attack has 65 power and is Flying type.
  – Headbutt has 70 power and is Normal type.
  – Return has 100 power and is Normal type.
  – Fire is powerful against Grass but weak to Water.
  – Grass is powerful against Water but weak to both Fire and Flying.
  – Water is powerful against Fire but weak to Grass.
  – Normal is not weak to anything but not powerful against anything either.
  – Flying is powerful against Grass and has no weaknesses.

• Write a query that returns all the Pokemon who can learn ‘Return’. (5)

• Write a query that returns all the moves in the game that are powerful against Grass. (5)

3. MySQL

Designing a database using MySQL was a smooth process. The initial request for step-by-step instructions to create five tables for the Pokemon problem statement was clear and effective. With the guidance provided, I successfully established the required tables: Pokemon, Type, Move, PokemonMove, and TypeEffectiveness.

Subsequently, when prompted individually, ChatGPT efficiently generated all the commands necessary for the assignment. The queries for both questions were accurately produced on the first attempt, demonstrating ChatGPT's effectiveness in generating SQL commands.

4. NoSQL-MongoDB

Despite my lack of prior experience with NoSQL, I expected more effective guidance from ChatGPT. However, encountering confusion prompted me to consult documentation. Ultimately, I successfully utilized MongoDB Compass and the Mongosh terminal to create the database and its five collections. Although this process proved to be a valuable learning experience in NoSQL database design, I found ChatGPT's assistance to be less helpful than I had anticipated.

5. Analysis and Conclusion

While MySQL database design was straightforward, thanks to clear step-by-step instructions, navigating NoSQL with MongoDB posed challenges due to inadequate guidance, necessitating additional effort and reliance on documentation. Despite these obstacles, the experience presented valuable learning opportunities in the realm of database design and management.

6. SQL Database Design

We'll design the database schema for the Pokemon system with the following tables:

6.1 Tables

1. Pokemon:

• id (PK): Unique identifier for each Pokemon.

• name: Name of the Pokemon.

• primary type id (FK to Type): Foreign key referencing the primary type of the Pokemon.

• secondary type id (FK to Type): Foreign key referencing the secondary type of the Pokemon (if

any).

2. Type:

• id (PK): Unique identifier for each type.

• name: Name of the type (e.g., Grass, Fire, Water).

3. Move:

• id (PK): Unique identifier for each move.

• name: Name of the move.

• power: Power of the move.

• type id (FK to Type): Foreign key referencing the type of the move.

4. Pokemon Move:

• pokemon id (FK to Pokemon): Foreign key referencing the Pokemon.

• move id (FK to Move): Foreign key referencing the move.

1

6.2 Populating the Tables

We'll populate the tables with the provided information.

```sql
-- Population for Pokemon table
INSERT INTO Pokemon (name, primary_type_id, secondary_type_id)
VALUES
    ('Bulbasaur', 1, NULL),
    ('Charmander', 2, NULL),
    ('Squirtle', 3, NULL),
    ('Eevee', 4, NULL),
    ('Pidgey', 4, 5);

-- Population for Type table
INSERT INTO Type (name)
VALUES
    ('Grass'),
    ('Fire'),
    ('Water'),
    ('Normal'),
    ('Flying');

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

-- Population for Pokemon_Move table
INSERT INTO Pokemon_Move (pokemon_id, move_id)
VALUES
    (1, 1),
    (1, 2),
    (1, 3),
    (2, 1),
    (2, 4),
    (2, 3),
    (3, 1),
    (3, 5),
    (3, 3),
    (4, 1),
    (4, 7),
    (4, 3),
    (5, 1),
    (5, 6),
    (5, 3);
```

Listing 1: SQL Table Population

6.3 Queries

We'll write queries to retrieve information from the database.

6.3.1 Query 1: Pokemon Learning 'Return'

```sql
SELECT p.name
FROM Pokemon p
JOIN Pokemon_Move pm ON p.id = pm.pokemon_id
JOIN Move m ON pm.move_id = m.id
WHERE m.name = 'Return';
```

Listing 2: Query to retrieve Pokemon learning 'Return'

6.3.2 Query 2: Moves Powerful Against Grass

```sql
SELECT m.name
FROM Move m
JOIN Type t ON m.type_id = t.id
WHERE t.name IN ('Fire', 'Flying') OR (t.name = 'Water' AND m.name != 'Water Gun');
```

Listing 3: Query to retrieve moves powerful against Grass