NOSQL IMPLEMENTATION

In Battle Royale Video Game

Abstract

Implementation of NoSQL database in Battle Royale video game

Badiganti, Karthik

kbadigan@kent.edu

NoSQL in Battle Royale Video Game

Scenario:

An independent gaming company wants to release a new game on multiple platforms like PC, console, and mobile. This game is a multiplayer battle royale game with a wide range of features and, characters and each character have its unique abilities, features, and attributes. There will be several options to select game fields and add-ons to the game. Also, in addition to the core game, it has plans to release several updates, and game packs, and based on promotional content it wants to add additional features to the game and update the game regularly with new content.

The company wants to store all the information and data about all these characters. The company's main aim is to handle the forthcoming user database more efficiently and quickly. To manage these data, it needs a flexible database where it can easily modify and update specific characters and apply additional game packs. They also want to be able to quickly search and retrieve specific characters and their attributes, as well as track player progress and statistics. For these reasons, the company is searching for a database solution.

Justification for NoSQL Solution:

The game has a large and rapidly growing player base: NoSQL databases are known for their ability to scale horizontally, which means that they can handle large amounts of data and a high volume of read and write requests without slowing down. This could be useful for a game with a large and rapidly growing player base, as it would allow the database to keep up with the demand.

The game has a lot of unstructured or semi-structured data: NoSQL databases are designed to handle large amounts of unstructured or semi-structured data, which could be useful for a game that has a lot of character data with different attributes and abilities.

The game needs to be able to store data in different formats: NoSQL databases allow you to store data in a variety of formats, including documents, key-value pairs, and graphs. This could be useful for a game that needs to store data about characters and their attributes in a flexible way, as it would allow the game developers to store the data in the format that best fits the needs of the game.

The game needs to be able to quickly search and retrieve data: NoSQL databases are known for their fast search and retrieval capabilities, which could be useful for a game that needs to be able to quickly search and retrieve data about specific characters and their attributes.

NoSQL DB Form:

The company wants to choose a database where it can handle large amounts of concurrency, is unstructured, and is flexible to update information about the game. For this purpose, the available options for NoSQL databases are column-oriented, key-value, and document-based databases.

Column Based: It is a distributed database designed for handling large amounts of data across multiple servers. While it could be a good choice for a game with a large and rapidly growing player base, it may not be the best fit for a game with a lot of unstructured or semi-structured data, as it is primarily designed for storing structured data.[i]

Key Value: It is an in-memory database that is known for its fast read and writes speeds. While it could be a good choice for a game that needs to be able to quickly search and retrieve data about

specific characters and their attributes, it may not be the best fit for a game with a large and rapidly growing player base, as it is primarily designed for storing small amounts of data in memory.[ii]

Document-Oriented: One of the main benefits of using a document-oriented database is storing and retrieving data in a flexible, hierarchical structure. Each character in the game could be represented as a document, with all their attributes, abilities, and appearance stored within that document. This would allow the studio to easily add and modify character data as needed, without having to worry about the rigid structure of a traditional relational database.[iii]

Another benefit of using a document-oriented database is the ability to perform efficient searches and queries. One such document-oriented database is MongoDB. It is highly scalable and can handle large volumes of data without experiencing performance issues. This would be especially important as the game grows in popularity and the studio needs to store and manage more data. The cost is very less when compared to other databases.

Implementation of Barebones Prototype:

A database is created in MongoDB using the company's game name. As a prototype, we will be creating a few collections and inserting documents in them. The database consists of User information, characters, and gameplay types.

This can be seen from the below snapshot. We have created a database called 'Kings of Valor' and then created three collections in them. They are,

1. Characters: This has the character information of the game
2. Game Types: This has different gameplay options available in the game.
3. Users: This collection has all the user information in them.

```
Atlas atlas-rx367a-shard-0 [primary] test1> use KINGS_of_VALOR
switched to db KINGS_of_VALOR
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.createCollection('users')
{ ok: 1 }
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.createCollection('characters')
{ ok: 1 }
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.createCollection('game_types')
{ ok: 1 }
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> show collections
characters
game_types
users
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR>
```

Below is the snapshot where the 'Users' collection is inserted with two documents where one user has a state field in it and the other doesn't have any state field in it. This is one example of unstructured data. It can also be observed that there are different data types are present in each document.

```
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.users.find().pretty()
[
  {
    _id: ObjectId("639d03f25ce0ceefd57946e5"),
    username: 'user12342',
    login_platform: [ 'PC', 'Mobile' ],
    play_time_hours: 670,
    personal_info: { mobile: '4324835358', state: 'OH' }
  },
  {
    _id: ObjectId("639d04d15ce0ceefd57946e6"),
    username: 'user34532',
    login_platform: [ 'PC', 'console' ],
    personal_info: { mobile: '231321324', state: 'CA', country: 'USA' },
    play_time_hours: 770
  }
]
```

The ‘Characters’ collection has documents related to characters in the game. Below are the two documents that are inserted each with different in them.

```
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.characters.find().pretty()
[
  {
    _id: ObjectId("639d08325ce0ceefd57946e7"),
    character: 'hero',
    combat_type: 'Aggressive',
    users_count: 311122,
    Abilities: { agility: 50, speed: 40, power: 60, strength: 70 }
  },
  {
    _id: ObjectId("639d08b45ce0ceefd57946e8"),
    character: 'Valkyrie',
    combat_type: 'Healer',
    users_count: 23122,
    Abilities: { agility: 30, speed: 40, heal_power: 60, Health: 70 }
  }
]
```

Below are the two documents that are inserted into the ‘Game Types’ collection.

```
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.game_types.find().pretty()
[
  {
    _id: ObjectId("639d0ac35ce0ceefd57946e9"),
    type: 'Arcade',
    duration_in_minutes: 30,
    max_players: 4,
    max_points: 40
  },
  {
    _id: ObjectId("639d0ae65ce0ceefd57946ea"),
    type: 'Battle Royale',
    duration_in_minutes: 90,
    max_players: 100,
    max_points: 99
  }
]
```

With this, our basic database is developed, and a few documents are inserted into the above-mentioned collections.

Functionality:

If the company wants to introduce a new character into the game, it can simply add it to the character’s collection without disturbing other collections. It can be observed below in the snapshot.

```
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.characters.find().pretty()
[
  {
    _id: ObjectId("639d08325ce0ceefd57946e7"),
    character: 'hero',
    combat_type: 'Aggressive',
    users_count: 311122,
    Abilities: { agility: 50, speed: 40, power: 60, strength: 70 }
  },
  {
    _id: ObjectId("639d08b45ce0ceefd57946e8"),
    character: 'Valkyrie',
    combat_type: 'Healer',
    users_count: 23122,
    Abilities: { agility: 30, speed: 40, heal_power: 60, Health: 70 }
  },
  {
    _id: ObjectId("639d0c8c5ce0ceefd57946eb"),
    character: 'Barbarian',
    combat_type: 'Warrior',
    Abilities: { agility: 60, speed: 90, Health: 80 }
  }
]
```

It can be observed that there is no 'user count' field in the newly added document character 'barbarian'. This could one example where data can be inserted more flexibly without any wastage of memory.

Consider another use case where the company wants to reduce the playtime of the Arcade game type and increase the players' count in that. It is implemented as below,

```
Atlas atlas-rx367a-shard-0 [primary] KINGS_of_VALOR> db.game_types.find().pretty()
[
  {
    _id: ObjectId("639d0ac35ce0ceefd57946e9"),
    type: 'Arcade',
    duration_in_minutes: 15,
    max_players: 8,
    max_points: 40
  },
```

The transactions which are performed above when implemented on a larger scale, the traditional database won't be able to handle them. MongoDB can efficiently update, delete, and insert documents and handle large-scale data.

References:

[i] https://www.integrate.io/blog/whats-unique-about-a-columnar-database/

[ii] https://severalnines.com/blog/intro-key-value-stores/

[iii] https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb/