УДК 519.8

ПЕРЕВАГИ NoSQL БАЗ ДАНИХ ДЛЯ МОНОФУНКЦІОНАЛЬНИХ СОЦІАЛЬНИХ СЕРВІСІВ.

Жилін М.О., студент, zhylin.8895345@stud.op.edu.ua
Вичужанін В. В., професор, Національний університет «Одеська політехніка».

Під час розробки будь-якого сервісу або застосунку неминуче постає питання, яку базу даних краще обрати. Тобто яка база даних буде оптимальною саме для цього сервісу або застосунку. Під час побудови свого сервісу для пошуку людей зі спільними інтересами, я зіткнувся з тим, що використання нереляційних баз даних (NoSQL) може бути зручнішим за реляційні (SQL) в моєму випадку.

Зараз дуже активно розвиваються різні соціальні сервіси – це можуть бути як невеликі вузькоспрямовані проекти від невеликих команд розробників, так і справжні мастодонти від великих компаній. Їх створюють цілячись у сучасну аудиторію, яка активно користується соціальними мережами і легко приймає нові корисні можливості сервісів, що народжуються. Це особливо актуально в сучасній Україні, адже допомагає легше взаємодіяти людям з якимись не поширеними хобі, знижуючи соціальну напруженість, а також допомагаючи підтримувати мораль у такий важкий час, сповнений стресу та психологічних випробувань.

На початковому етапі створення такого проекту, в якому різні люди зі своїми захопленнями могли б взаємодіяти і легко знаходити один одного, я зіткнувся з проблемою вибору того, які ж технології будуть найкращими в моєму варіанті розробки та створення сервісу з великим обсягом крос-секційних даних.

І найголовніша проблема, з якою я зіткнувся, це вибір типу бази даних. Сучасних систем баз даних є безліч, і на початку видавалося природним використовувати SQL систему. У моєму проекті передбачається, що в кожного користувача є свій профіль, який він налаштовує, а також деякі картки зацікавленостей та спільнот, у яких, в свою чергу, є додаткові параметри, специфічні саме для того типу хобі, яке вказав користувач. Це можуть бути хобі, пов'язані з комп'ютерними іграми, де є великі команди, і таким чином людина знаходить, з ким спілкуватися. Або, наприклад, з настільними іграми, задля яких можна збиратися великою компанією друзів.

В такому випадку ми отримуємо велику кількість різних параметрів і об'єктів, вкладених один в одного. При використанні класичної SQL системи, довелося б створювати багатотабличну систему з великою кількістю полів, а також заздалегідь продумувати і створювати визначення того, які саме параметри важливіші, що накладає деякі обмеження на кінцеву реалізацію і можливості сервісу [1].

При побудові саме SQL баз даних, можна було б реалізувати це через зв'язки декількох таблиць, де Таблиця 1 зберігала б базові дані користувача, Таблиця 2 зберігала б дані про типи хобі і Таблиця 3 компонувала б ці дані, щоб можна було реалізувати потрібну нам можливість вказувати велику кількість захоплень користувача. На мій погляд, це не дуже зручний варіант, і він все одно залишає вимогу суворої типізації карток хобі.

Таблиця 1 – Приклад таблиці для даних користувача.

ID	Name	Surname	City
CHAR(10) NOT NULL Primary key,	CHAR(20) NOT NULL	CHAR(20) NOT NULL	CHAR(50) NOT NULL
78	Сергій	Базетов	Київ

Таблиця 2 – Приклад таблиці для даних карток.

ID	Title	Description
CHAR(10) NOT NULL Primary key,	CHAR(20) NOT NULL	CHAR(200) NOT NULL
57	Манчкін	Настільна карткова гра

Причому розмір Таблиці 3 був би дуже великим внаслідок того, що кожен зв'язок облікового запису користувача з карткою захоплення був би новим записом. Авжеж є можливість зробити це як зберігання ID хобі користувача у формі масиву, а цей масив зберігати у вигляді текстового рядка, додаючи спеціальний символ-розділювач між елементами масиву. Наприклад, масив з трьох елементів зберігати як “120 78 985” у вигляді “120&78&985”. При записі ми об'єднуємо масив в один рядок і записуємо його в БД, при читанні ми читаємо рядок і ділимо рядок на елементи по роздільнику. На мій погляд, це створює надто великі незручності у розробці.

Таблиця 3 – Приклад таблиці для даних зв’язків.

ID	ID User	ID Pastimes
CHAR(10) NOT NULL Primary key,	CHAR(20) NOT NULL	CHAR(20) NOT NULL
1	78	57

Саме тому я звернув увагу на NoSQL системи. Вони працюють за принципом JSON-подібних документів, що відкриває велику кількість можливостей для мого проекту. А саме, я створюю форму з основними ключами, які заповнює користувач, стандартного типу, такі як ім'я, місто за бажанням, вік і так далі. Після цього останнім параметром я вказую параметр "хобі", куди і будуть зберігатися картки з інформацією про те, чим саме захоплюється користувач [2]. Ці картки мають шаблони під різні базові типи хобі, але за потреби можна додавати необмежену кількість підпараметрів. При цьому користувач може зробити декілька таких карток для різних хобі, усі вони зберігаються у форматі, схожому на асоціативний масив. Виходить так, що в одному зазначеному нами параметрі "хобі" зберігатиметься різна кількість карток зі своєю інформацією, також вони можуть зберігати в собі, наприклад, посилання на деякі групи, якщо такий функціонал знадобиться.

```
{
        "_id" : "5dd4e2cc0821d3b44607534c",
        "name" : "Сергій",
        "surname" : "Базетов",
        "city" : "Київ",
        "pastimes" : [
            {
                "id" : "6gfhhgj4bvnc5567cghjmnbm",
                "title" : "Манчкін",
                "description" : "Настільна карткова гра."
            },
            {
                "id" : "l256k2kn2bmkj456hilds2fl",
                "title" : "Футбол",
                "description" : "Фанат Динамо"
            }

        ]
}
```


Рисунок 1 – Приклад документу з даними у JSON форматі

Таким чином, це значно спрощує розробку, особливо за рахунок того, що нам не потрібно використовувати SQL код, який би довелося робити для оптимізації запитів при створенні даного сервісу на звичайних SQL базах даних. Все, що нам потрібно, це вказати базові параметри, ідентичні для всіх користувачів, а потім вказати один параметр, який буде зберігати масиви, які ми вже створюємо окремо в інтерфейсі користувача, де він може їх налаштовувати, заповнювати, вказувати додаткову інформацію.

Це доповнює та збільшує варіативність використання нашого сервісу та зручність у його розробці. Звичайно, присутні й певні мінуси даного підходу. Наприклад, якщо ми зробимо так, щоб картки, які зберігають інформацію про

якісь хобі, користувач міг брати у різних груп, які будуть таким чином спільними для великої кількості користувачів, тоді даний спосіб не дозволяє завжди використовувати для відображення актуальну інформацію, адже цілком може статися, що вихідний об'єкт змінив назву або опис, внаслідок чого збережені дані вимагають актуалізації. Її можна реалізувати на боці вихідного об'єкта, який при зміні братиме всі пов'язані з ним документи користувачів і оновлювати там дані, що є кращим, але витратним методом.

Висновок: Можна сказати, що використання нереляційних баз даних при побудові монофункціональних соціальних сервісів, які використовують безліч різних даних, які можуть мати різні параметри і які при цьому не вимагають складних обчислювальних операцій над ними, може бути кращим варіантом. Це було продемонстровано на прикладі такої бази даних для сервісу, що потребує збереження змінної кількості даних для однієї моделі.

Список використаних джерел

1. Michael Kaufmann, Andreas Meier. Database Modeling. SQL and NoSQL Databases; 2023: 25 – 67.

2. Bing Li, Juan Wang, Ning Li, Minghua Zhao. Application and Development of Database Technology in the Background of Big Data. Advances in Computer Science Research, № 87; 2019: 947 – 948.