Лабораторна робота № 2. СУБД типу “Ключ-Значення”: Redis

МЕТА РОБОТИ: отримати/закріпити навички роботи з базами даних типу “Ключ-Значення”.

ПОСТАНОВКА ЗАДАЧІ

Реалізувати клієнтський застосунок, що буде здійснювати запити до бази даних, яка була розроблена у Лабораторній роботі № 1. Для кешування запитів до бази даних використати СУБД Redis (рис. 1).

Рівень 1 (середній). Обидві СУБД (MongoDB та Redis) повинні знаходитись на тій же робочій машині, що і сам застосунок. Оцінюється у **8 балів**.

Рівень 2 (підвищений). Базу даних, що була розроблена в ЛР № 1 попередньо перенести в хмарну СУБД MongoDB Atlas. Redis знаходиться на тій же машині, що й клієнтський застосунок. Оцінюється у **10 балів**.

Примітки! Мова програмування (для обох рівнів), на якій буде розроблено клієнтський застосунок, обирається студентом самостійно.

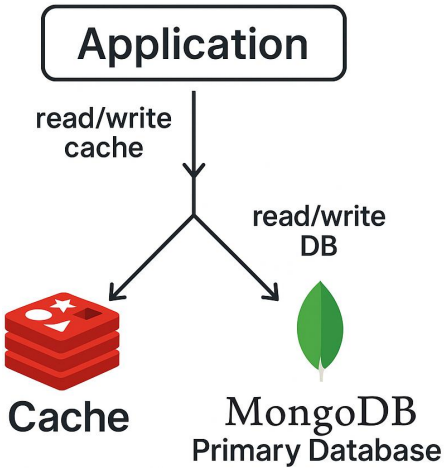

Рис. 1. Архітектура застосунку

ПОРЯДОК ВИКОНАННЯ

1. Встановити СУБД Redis відповідно до інструкції з офіційного сайту https://redis.io/docs/latest/operate/oss_and_stack/install/archive/install-redis/

2. (Опціонально) Якщо ви не виконували лабораторну роботу по Redis з курсу Бази даних, ознайомтесь з основними типами даних Redis, використовуючи офіційну документацію, лекційний матеріал, інші навчальні матеріали з переліку джерел до цього курсу. Наведіть скріни, що демонструють результати виконання

команд СУБД Redis у звіті. Для виконання команд можна скористатись або redis-cli або онлайн терміналом https://onecompiler.com/redis.

3. Розробити клієнтський застосунок, в якому реалізувати запити на додавання, пошук та видалення інформації з БД, що була розроблена у Лабораторній роботі № 1.

4. Додати до застосунку рівень кешування даних. Для реалізації кешування обрати одну з основних двох стратегій кешування даних:

1) ache-aside (для читання) + write-around (для зипису),
2) cache-aside (для читання) + write-through (для запису).

Стратегію кешування обрати в залежності від номера за списком групи, непарні номери обирають першу стратегію, парні − другу. Для зберігання закешованих даних використати СУБД Redis. В обох варіантах налаштовувати час існування данних у кеші (через TTL - Time To Live).

5. Заміряти час, яких потрібний для виконання запиту для 2-х випадків:

1) коли дані читаються з основної бази даних;
2) коли читаються закешовані дані.

Порівняйте отримані результати.

Зверніть увагу! У випадку виконання завдання з **основного рівня** ефект від кешування на замірах часу може бути непомітний, оскільки документоорієнтовані СУБД самі по собі досить швидкі. Наступні поради можуть в цьому дещо допомогти, але не гарантовано. Ефект був помітний, якщо в базі даних буде досить багато даних (10 000, 100 000 і більше), особливо за умови складних аналітичних запитів, як це і має місце в реальних сценаріях використання MongoDB (як основної СУБД) та Redis (для кешування).

- Оскільки MongoDB сама кешує плани виконання запитів, рекомендується очищати власний кеш планів запитів у MongoDB за допомогою команди planCacheClear();
- Рекомендується також заміри часу виконання запитів робити при пошуку даних, коли не будуть використовуватись створені індекси, що пришвидшують пошук;
- Для замірів часу виконувати запити на пошук даних з використанням агрегатних функцій;
- Якщо надані поради не допоможуть, скористатись традиційним засобом імітації складної логіки :-): функцією sleep().

6. У звіті навести код застосунку та отримані результати замірів часу виконання запитів.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. До якого типу NoSQL СУБД відноситься Redis? Які ще СУБД з цього класу є популярними на поточний момент? Аргументуйте свою відповідь, прикріпивши посилання на відповідні рейтинги.
2. Які типи даних пропонує для використання СУБД Redis?
3. Які типові варіанти застосування БД на базі СУБД Redis?
4. З чим пов'язана досить висока швидкість виконання CRUD-операцій в Redis у порівнянні з іншими СУБД?
5. Що таке реплікація даних? Яка її мета?
6. Який механізм реплікації даних використовується в Redis?
7. Що таке шардинг? Яка його мета?
8. Чим реплікація відрізняється від шардингу?
9. Які стратегії кешування даних ви знаєте? Які з них частіше використовуються і в яких сценаіях?

Приклад використання СУБД Redis для кешування запитів до MongoDB на мові Python. У якосту бази даних використано навчальну базу даних learn (колекція unicorns), яка присутня в MongoDB після встановлення цієї СУБД. В прикладі відсутня реалізація логіки замірів часу виконання запитів.

```
import redis
import pymongo
import json
from bson.objectid import ObjectId

mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")
mongo_db = mongo_client["learn"]
unicorns_col = mongo_db["unicorns"]
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# CACHE-ON READ
def get_unicorn_by_name(name):
    cache_key = f"unicorn:{name}"
    cached_data = redis_client.get(cache_key)

    if cached_data:
        return json.loads(cached_data)

    # Not in cache, fetch from MongoDB and update cache
    unicorn = unicorns_col.find_one({"name": name}, {"_id": 0})
    if unicorn:
        # Before caching, convert the ObjectId to a string, as JSON doesn't know how to serialize ObjectId.
        if '_id' in unicorn and isinstance(unicorn['_id'], ObjectId):
            unicorn['_id'] = str(unicorn['_id'])
        redis_client.set(cache_key, json.dumps(unicorn), ex=60)  # cache for 60 seconds
```

```
    return unicorn

# CASHIING ON WRITE
def add_unicorn(unicorn_data):
    name = unicorn_data["name"]
    unicorns_col.insert_one(unicorn_data)  # Insert into MongoDB

    # Before caching, convert the ObjectId to a string, as JSON doesn't know how to serialize ObjectId.
    if '_id' in unicorn_data and isinstance(unicorn_data['_id'], ObjectId):
        unicorn_data['_id'] = str(unicorn_data['_id'])
    # Immediately update cache
    cache_key = f"unicorn:{name}"
    redis_client.set(cache_key, json.dumps(unicorn_data), ex=60)

if __name__ == "__main__":
    try:
        redis_client.ping()
        mongo_client.admin.command('ping')

        unicorns_col.delete_many({"name": "Starlight"})  # Clean up collection to avoid duplicates
        redis_client.delete("unicorn:Starlight")  # Clearing the cache for Starlight
        new_unicorn = {
            "name": "Starlight",
            "gender": "f",
            "age": 5,
            "loves": ["adventure", "stars"],
            "weight": 420
        }

        add_unicorn(new_unicorn)
        # Some logic ...
        result = get_unicorn_by_name("Starlight")  # data is retrieved from cache
        print("Result:", result)

    except redis.exceptions.ConnectionError as e:
        print(f"ERROR: Could not connect to Redis. Error: {e}")
    except pymongo.errors.ConnectionFailure as e:
        print(f"ERROR: Could not connect to MongoDB. Error: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```