Лабораторна робота №9

З дисципліни: Бази даних та інформаційні системи

Студента групи МІТ-31: Заяць Діани

Тема: Розширені можливості Redis

Мета: Закріпити знання про роботу з Redis та ознайомитися з розширеним функціоналом — транзакціями, скриптами на Lua, публікацією/підпискою (Pub/Sub) та потоками Redis Streams.

Хід роботи:

1. **Транзакції у Redis**

У Redis транзакції дають змогу виконувати декілька команд в рамках одного атомарного блоку. Основні команди для роботи з транзакціями:

- **MULTI** — початок транзакції.
- **EXEC** — виконання всіх команд, які були викликані після MULTI.
- **DISCARD** — скасування транзакції.
- **WATCH** — спостереження за ключами, якщо значення змінюється під час транзакції, то транзакція не виконується.

а) Прості транзакції

1. Встановимо баланс і кількість покупок у Redis
2. Потім створимо транзакцію для зменшення балансу на 10 і збільшення кількості покупок на 1.

Перевіряємо результати:

```
127.0.0.1:6379> SET balance 100
OK
127.0.0.1:6379> SET purchases 0
OK
127.0.0.1:6379> WATCH balance
OK
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379(TX)> DECR balance
QUEUED
127.0.0.1:6379(TX)> INCR purchases
QUEUED
127.0.0.1:6379(TX)> EXEC
1) (integer) 99
2) (integer) 1
127.0.0.1:6379> GET balance
"99"
127.0.0.1:6379> GET purchases
"1"
127.0.0.1:6379>
```

b) Транзакція з кількома ключами

1. Створимо транзакцію, яка додає кілька ключів:

```
127.0.0.1:6379(TX)> SET key1 "value1"
QUEUED
127.0.0.1:6379(TX)> SET key2 "value2"
QUEUED
127.0.0.1:6379(TX)> EXEC
1) OK
2) OK
127.0.0.1:6379> GET key1
"value1"
127.0.0.1:6379> GET key2
"value2"
127.0.0.1:6379>
```

c) Імітація конкурентного доступу за допомогою `WATCH`

1. Спостерігаємо за ключем `balance`, потім виконуємо транзакцію з іншими командами. Якщо інший клієнт змінить значення `balance`, то транзакція не буде виконана.

```
127.0.0.1:6379> WATCH balance
127.0.0.1:6379(TX)> DECR balance
QUEUED
127.0.0.1:6379(TX)> INCR purchases
QUEUED
127.0.0.1:6379(TX)> EXEC
1) (integer) 98
2) (integer) 2
"value"
```

2. Lua-скрипти в Redis

Команда **`EVAL`** дозволяє виконувати Lua-скрипти в Redis.

a) Lua-скрипт для перевірки наявності ключа

1. Напишемо Lua-скрипт, який перевіряє наявність ключа і створює його, якщо він не існує і перевіримо значення ключа

```
127.0.0.1:6379> EVAL "if redis.call('exists', KEYS[1]) == 0 then return re
dis.call('set', KEYS[1], ARGV[1]) else return 'exists' end" 1 testkey "val
ue"
"exists"
127.0.0.1:6379> GET testkey
"value"
127.0.0.1:6379>
```

3. Механізм Pub/Sub

Redis підтримує модель **Pub/Sub** (публікація/підписка), яка дозволяє одному процесу публікувати повідомлення, а іншому процесу підписуватися на ці повідомлення.

a) Запуск підписки на канал

1. В одному терміналі запустимо підписку на канал `news`:

```
127.0.0.1:6379> SUBSCRIBE news
```

b) Публікація повідомлення на канал

2. В іншому терміналі публікуємо повідомлення на канал `news`:

```
127.0.0.1:6379> PUBLISH news "Redis is awesome!"
(integer) 1
127.0.0.1:6379>
```

Підписка отримує повідомлення і виводить на еран:

```
1) "subscribe"
2) "news"
3) (integer) 1
1) "message"
2) "news"
3) "Redis is awesome!"
Reading messages... (press Ctrl-C to quit
```

4. Redis Streams (Потоки даних)

Redis Streams дозволяє обробляти події в реальному часі, як це роблять черги повідомлень.

а) Додавання подій до потоку

1. Додамо подію до потоку `mystream` і перевіримо події
2. Створимо споживача потоку, який зчитує нові повідомлення з потоку `mystream`

```
127.0.0.1:6379> XADD mystream * sensor-id 1234 temperature 19.8
"1746690152627-0"
127.0.0.1:6379> XRANGE mystream - +
1) 1) "1746690152627-0"
   2) 1) "sensor-id"
      2) "1234"
      3) "temperature"
      4) "19.8"
127.0.0.1:6379> XREAD COUNT 2 STREAMS mystream 0
1) 1) "mystream"
   2) 1) 1) "1746690152627-0"
         2) 1) "sensor-id"
            2) "1234"
            3) "temperature"
            4) "19.8"
127.0.0.1:6379>
```

Додаткове завдання

```python
import redis
import threading
import time

class CounterSystem:
    def __init__(self):
        self.r = redis.Redis(host='localhost', port=6379, db=0)
        self.channel = 'event_channel'  # Канал для Pub/Sub
        self.stream_name = 'event_stream'  # Потік для зберігання подій

    def listen_for_events(self):
        """Функція для підписки на канал і отримання подій"""
        pubsub = self.r.pubsub()
```

```
        pubsub.subscribe(self.channel)

        print("Підписка на канал... Чекаю подій.")
        for message in pubsub.listen():
            if message['type'] == 'message':
                print(f"Нова подія: {message['data'].decode('utf-8')}")

    def add_event(self, event_name, count):
        """Функція для додавання події в канал і потік"""
        try:
            count = int(count)
        except ValueError:
            print("Кількість повинна бути числом!")
            return

        # Публікуємо подію в канал
        self.r.publish(self.channel, f"{event_name} - {count} new events")

        # Додаємо подію в потік, кодуємо значення в байти
        self.r.xadd(self.stream_name, {
            'event': event_name.encode('utf-8'),
            'count': str(count).encode('utf-8'),
            'timestamp': str(time.time()).encode('utf-8')
        })

    def show_event_history(self):
        """Функція для перегляду історії подій з потоку"""
        messages = self.r.xrange(self.stream_name, '-', '+')
        if not messages:
            print("Історія порожня.")
        else:
            print("\nІсторія подій:")
            for message in messages:
                message_id = message[0]  # перший елемент - ID події
                message_data = message[1]  # другий елемент - дані події
                # Перевірка наявності полів 'event' і 'count'
                event_name = message_data.get(b'event',
b'Unknown').decode('utf-8')
                count = message_data.get(b'count', b'0').decode('utf-8')
```

```
                timestamp = message_data.get(b'timestamp',
b'N/A').decode('utf-8')

                # Виведення події
                print(f"ID: {message_id.decode('utf-8')} | Подія:
{event_name} | Кількість: {count} | Час: {timestamp}")

# Функція для запуску підписки в окремому потоці
def start_listening(counter_system):
    listener_thread =
threading.Thread(target=counter_system.listen_for_events)
    listener_thread.daemon = True
    listener_thread.start()

# Основний блок
def main():
    counter_system = CounterSystem()

    # Запуск підписки на канал в окремому потоці
    start_listening(counter_system)

    while True:
        print("\n1. Додати подію")
        print("2. Переглянути історію подій")
        print("3. Вийти")

        choice = input("Виберіть опцію: ")

        if choice == '1':
            event_name = input("Введіть назву події: ")
            count = input("Введіть кількість нових подій: ")
            counter_system.add_event(event_name, count)
        elif choice == '2':
            counter_system.show_event_history()
        elif choice == '3':
            print("До побачення!")
            break
        else:
            print("Невірний вибір. Спробуйте ще раз.")
```

```python
if __name__ == "__main__":
    main()
```

Як працює:

```
1. Додати подію
2. Переглянути історію подій
3. Вийти
Виберіть опцію: Підписка на канал... Чекаю подій.
1
Введіть назву події: Weekend
Введіть кількість нових подій: 1
Нова подія: Weekend - 1 new events

1. Додати подію
2. Переглянути історію подій
3. Вийти
Виберіть опцію: 2

Історія подій:
ID: 1746690966888-0 | Подія: Monday | Кількість: 1 | Час: 1746690966.8889556
ID: 1746691034456-0 | Подія: monday | Кількість: 1 | Час: 1746691034.454966
ID: 1746691177860-0 | Подія: lalala | Кількість: 1 | Час: 1746691177.859875
ID: 1746691353971-0 | Подія: monday | Кількість: 1 | Час: 1746691353.9714694
ID: 1746691506343-0 | Подія: monday | Кількість: 1 | Час: 1746691506.342527
ID: 1746691570797-0 | Подія: hello world | Кількість: 1 | Час: 1746691570.7969308
ID: 1746693303045-0 | Подія: Weekend | Кількість: 1 | Час: 1746693303.0462954

1. Додати подію
2. Переглянути історію подій
3. Вийти
Виберіть опцію:
```