

CS200 FINAL: Installing and Using Redis to Create Databases

Anson Ekau, Mia Wong, Kandielynn Ueligitone, Christian Raquepo

School of Natural Sciences and Mathematics CS-200-01-1

Dr. Rylan Chong

December 1st, 2023

Introduction

According to IBM Redis is “an open source, in-memory, NoSQL key/value store that is used primarily as an application cache or quick-response database.” To put it in simpler terms, Redis allows users to store information and access them fast. To use an anecdote, Redis can be seen as a notebook where a user can write down information and at any time quickly access that information. Additionally a user can also organize what they store. This feature refers to the “key/value store” in the docker hub definition. Which will be discussed in depth later. Some additional key points from this definition is the fact that Redis is open-source, meaning anyone can contribute and use Redis. Redis is also “in-memory,” meaning that Redis operates on a network and stores information in a system's RAM. These are the key definitions for understanding how Redis works. The following section will breakdown these definitions in detail starting with the easier concepts to understand.

First, Redis is open-sourced. So anyone can contribute to the project and use it. There are advantages to an open-source project. According to Refine, one benefit of open source projects is the code quality. Open source projects have better quality code “because there are more eyes on the code and more people who are willing to contribute to fix issues.”

Second, Redis is stored in-memory, specially on the RAM of a system. This makes Redis more efficient as items stored on RAM are faster to access. Livewire uses the analogy of a desk to explain RAM, “Think of RAM like an office desk. A desk is used for quick access to important documents, writing tools, and other items that you need *right now*... This type of memory, like a desk in the analogy, provides much faster read/write times than using a hard drive.” Since RAM is more efficient, it allows Redis to access and edit information faster. These

are two characteristics that make Redis appealing to use. The speed and the ease of updating information.

Third, as mentioned earlier, Redis is a key/value data store. Recall the idea that Redis is like a notebook where you can write and access information quickly. The information can also be organized and this is the key-value data store feature. To understand this concept, let's build on the analogy of the notebook. You can organize the information (value) in your notebook with labels (keys). Say you want to access a specific piece of information (value), all you would need to do is query the specific label (key) assigned to the value. Additionally you can store different types of values. For example strings, numerics, hash and etc. As mentioned previously, accessing keys and values is fast, and updating them is too.

To recap this section, Redis is an open source, in-memory, key/value data store. Redis is characterized by its speed and ease of use. Redis relies on the system's RAM to carry out key functions pertaining to the key/value data store system Redis uses. In addition to this Redis is open source so anyone can use and contribute to the project. This characteristic brings advantages, one of them being better code quality.

In the following sections, this paper will delve deeper into the key features of Redis, provide a short instructional on installing and setting up Redis, and a tutorial on performing basic and common Redis code. The tutorials will specifically focus on the Mac version of Redis using python scripting in Jupyter Notebook.

Key Features of Redis compared to MySQL

	REDIS	MySQL
MEMORY STORAGE	Max: 300 GB	512MB of RAM
DATA STRUCTURES	<input type="checkbox"/> Strings <input type="checkbox"/> Sets <input type="checkbox"/> Lists <input type="checkbox"/> Hashes <input type="checkbox"/> Binary	<input type="checkbox"/> String <input type="checkbox"/> Numeric <input type="checkbox"/> date and time.
PERFORMANCE	<input type="checkbox"/> 1000000 requests completed in 7.11 seconds <input type="checkbox"/> 50 parallel clients <input type="checkbox"/> 3 bytes payload	<input type="checkbox"/> 1 row in set (0.0130 sec)
USE CASES	<input type="checkbox"/> supports application search <input type="checkbox"/> cache	<input type="checkbox"/> Social media <input type="checkbox"/> Government <input type="checkbox"/> Media and entertainment <input type="checkbox"/> Fraud detection <input type="checkbox"/> Business mapping <input type="checkbox"/> e-commerce

Installing Redis

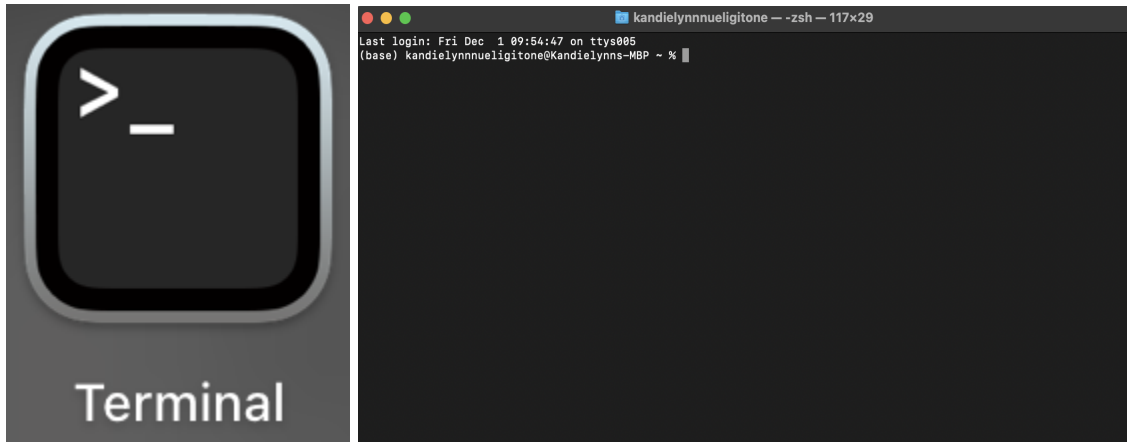
Installation for Mac:

Before installing Redis on Mac, Homebrew needs to be installed first. Homebrew is a package manager for macOS. It simplifies the installation of software on the Mac operating system. It installs packages to their own directory and then symlinks. It's not recommended to directly work within the Homebrew directory for your projects, as this directory is managed by Homebrew for installing and maintaining software packages. According to the source “Installing Redis on macOS,” Homebrew is the easiest way to install Redis on macOS.

Step 1: Install Homebrew

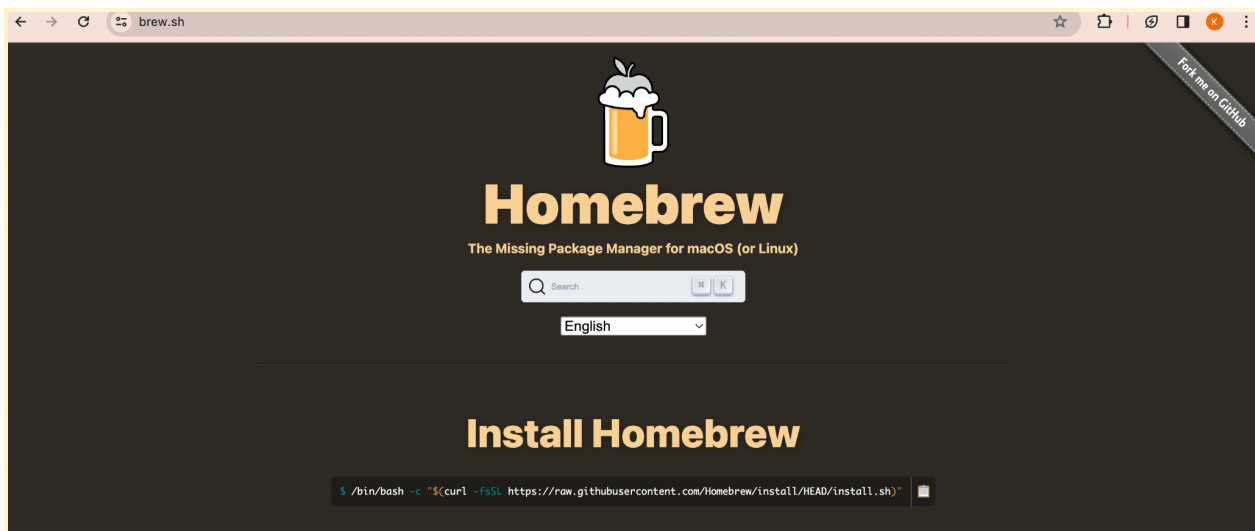
1. Open Terminal:

Location of Terminal on Mac: You can find the Terminal application in the Utilities folder in your Applications folder.



2. Install Homebrew: search <https://brew.sh/> and copy the command given on the webpage.
Paste the following command into your terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```



Once you run the code, follow the instructions provided in the output:

==> Next steps:

- Run these two commands in your terminal to add Homebrew to your **PATH**:
(echo; echo 'eval "\$(/opt/homebrew/bin/brew shellenv)"') >> /Users/kandielynnnueligitone/.zprofile
eval "\$(/opt/homebrew/bin/brew shellenv)"

Step 2: Install Redis

With Homebrew installed, you can easily install Redis.

1. Install Redis: Run this command to install Redis:

```
brew install redis
```

```
[(base) kandielynnnueligitone@Kandielynn-MBP ~ % brew install redis]

==> Downloading https://ghcr.io/v2/homebrew/core/redis/manifests/7.2.3
##### 100.0%
==> Fetching dependencies for redis: ca-certificates and openssl@3
==> Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/manifests/2023-08-22
##### 100.0%
==> Fetching ca-certificates
==> Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/blobs/sha256:a331e92e7a759571296581f029e5cc2ec7cee70cd92dc0b5f8eb76095f94a21a
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/3/manifests/3.1.4
##### 100.0%
==> Fetching openssl@3
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/3/blobs/sha256:81f024814ded9921e2d3d57702afb253a23c77530e5cfa17b04d6353436a428b
##### 100.0%
==> Fetching redis
==> Downloading https://ghcr.io/v2/homebrew/core/redis/blobs/sha256:f2bfbdb496e880913366a7725958e34d2a4dc75455777c2962c840a5c1705b76
##### 100.0%
==> Installing dependencies for redis: ca-certificates and openssl@3
==> Installing redis dependency: ca-certificates
==> Downloading https://ghcr.io/v2/homebrew/core/ca-certificates/manifests/2023-08-22
Already downloaded: /Users/kandielynnnueligitone/Library/Caches/Homebrew/downloads/a8cd32e30cae0b7335779e93a6554f294f09485082fc253a3a8be441337a6115--ca-certificates-2023-08-22.bottle_manifest.json
==> Pouring ca-certificates--2023-08-22.arm64_monterey.bottle.tar.gz
==> Regenerating CA certificate bundle from keychain, this may take a while...
  /opt/homebrew/Cellar/ca-certificates/2023-08-22: 3 files, 221.6KB
==> Installing redis dependency: openssl@3
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/3/manifests/3.1.4
Already downloaded: /Users/kandielynnnueligitone/Library/Caches/Homebrew/downloads/528fccab152aa1785e7e73f54f268d413bf94e22188c0e4ff17a6c497d2783b88--openssl@3-3.1.4.bottle_manifest.json
==> Pouring openssl@3--3.1.4.arm64_monterey.bottle.tar.gz
  /opt/homebrew/Cellar/openssl@3/3.1.4: 6,496 files, 28.4MB
==> Installing redis
==> Pouring redis--7.2.3.arm64_monterey.bottle.tar.gz
==> Caveats
To start redis now and restart at login:
brew services start redis
Or, if you don't want/need a background service you can just run:
/opt/homebrew/opt/redis/bin/redis-server /opt/homebrew/etc/redis.conf
==> Summary
  /opt/homebrew/Cellar/redis/7.2.3: 14 files, 2.5MB
==> Running 'brew cleanup redis'...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see 'man brew').
==> Caveats
==> redis
To start redis now and restart at login:
brew services start redis
Or, if you don't want/need a background service you can just run:
/opt/homebrew/opt/redis/bin/redis-server /opt/homebrew/etc/redis.conf
...
```

Step 3: Launch Redis

After installing Redis, you need to start it.

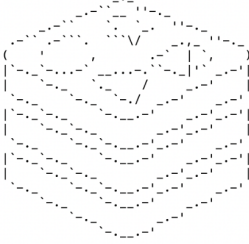
1. Start Redis: if you don't want/need a background service you can just run:

```
redis-server
```

```

Last login: Thu Nov 16 13:16:16 on ttys001
(base) kandielynnueligitone@Kandielynnns-MBP ~ % redis-server

72068:C 16 Nov 2023 13:18:54.172 * o00o000o000o Redis is starting o00o000o000o
72068:C 16 Nov 2023 13:18:54.172 * Redis version=7.2.3, bits=64, commit=00000000, modified=0, pid=72068, just started
72068:C 16 Nov 2023 13:18:54.172 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
72068:M 16 Nov 2023 13:18:54.173 * Increased maximum number of open files to 10032 (it was originally set to 2560).
72068:M 16 Nov 2023 13:18:54.173 * monotonic clock: POSIX clock_gettime



Redis 7.2.3 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 72068

https://redis.io

72068:M 16 Nov 2023 13:18:54.174 # WARNING: The TCP backlog setting of 511 cannot be enforced because kern.ipc.somaxconn is set to the lower value of 128.
72068:M 16 Nov 2023 13:18:54.175 * Server initialized
72068:M 16 Nov 2023 13:18:54.175 * Ready to accept connections tcp

```

2. In order to connect to Redis, open another terminal window and run ***redis-cli***:
3. Test Redis: To make sure Redis is working, open another terminal window and type:

ping

If Redis is running, it will return 'PONG'.

```

[127.0.0.1:6379> ping
PONG

```

Step 4: Connect Redis to Python on Terminal

To interact with Redis from your Jupyter Notebook, you need to install a Redis client for Python.

To connect Redis to Python, run this command:

pip install redis

```

(base) kandielynnueligitone@Kandielynnns-MBP ~ % pip install redis

Collecting redis
  Downloading redis-5.0.1-py3-none-any.whl (250 kB)
    250.3/250.3 kB 1.2 MB/s eta 0:00:00
Collecting async-timeout>=4.0.2
  Downloading async_timeout-4.0.3-py3-none-any.whl (5.7 kB)
Installing collected packages: async-timeout, redis
Successfully installed async-timeout-4.0.3 redis-5.0.1

```

Using Redis on Python (Jupyter Notebook):

Open Anaconda and launch Jupyter Notebook.

1. ***Connect to Redis in Your Notebook:*** Once the Redis server is running and you have the Redis client installed in your notebook, you can connect to Redis using Python code.

Here's a basic example to get you started:

```
import redis
```

```
# Connect to Redis
```

```
r = redis.Redis(host='localhost', port=6379, db=0)
```

```
# Set a key
```

```
r.set('test_key', 'Hello, Redis!')
```

```
# Get the value back
```

```
print(r.get('test_key')) # Should output: b'Hello, Redis!'
```

Output Result:

```
b'Hello, Redis!'
```


Coding with Redis

A. Creating a Database or Starting Application.

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Set a key-value pair
redis_client.set('my_key', 'my_value')

# Get the value for a given key
value = redis_client.get('my_key')
print('Value for key "my_key":', value)

# Increment a value
redis_client.incr('counter_key')
counter_value = redis_client.get('counter_key')
print('Counter value:', counter_value)
```

Value for key "my_key": my_value
Counter value: 1

*The code is in the gray box and the output is underneath.

- **Connecting to Redis:**

`redis.StrictRedis` is a class provided by the `redis-py` library to interact with a Redis server.

You need to provide the host, port, and optionally the database number. In this example, it connects to the default database (`db=0`).

- **Setting and Getting Values:**

You can use `set` to set a key-value pair and `get` to retrieve the value for a given key.

- **Incrementing a Value:**

The `incr` command is used to increment the value of a key. If the key doesn't exist, it will be initialized with the value of 1.

B. Creating Tables

Code Example for Creating 3 Tables

User Information Table: Simulated using Redis hashes.

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Example 1: User Information Table (Using Hash)
def add_user(user_id, username, email):
    user_key = f"user:{user_id}"
    redis_client.hset(user_key, "user_id", user_id)
    redis_client.hset(user_key, "username", username)
    redis_client.hset(user_key, "email", email)

def get_user(user_id):
    user_key = f"user:{user_id}"
    user_info = redis_client.hgetall(user_key)
    return user_info
```

List of Recent Activities Table: Simulated using Redis lists.

```
# Example 2: List of Recent Activities Table (Using List)
def log_recent_activity(user_id, activity):
    recent_activities_key = f"recent_activities:{user_id}"
    redis_client.lpush(recent_activities_key, activity)
    redis_client.ltrim(recent_activities_key, 0, 4) # Keep only the latest 5 activities

def get_recent_activities(user_id):
    recent_activities_key = f"recent_activities:{user_id}"
    activities = redis_client.lrange(recent_activities_key, 0, -1)
    return activities
```

Set of Followers Table: Simulated using Redis sets.

```
# Example 3: Set of Followers Table (Using Set)
def add_follower(user_id, follower_id):
    followers_key = f"followers:{user_id}"
    redis_client.sadd(followers_key, follower_id)

def get_followers(user_id):
    followers_key = f"followers:{user_id}"
    followers = redis_client.smembers(followers_key)
    return followers

# Example Usage
add_user("1", "john_doe", "john@example.com")
log_recent_activity("1", "Logged in")
add_follower("1", "2")

user_info = get_user("1")
recent_activities = get_recent_activities("1")
followers = get_followers("1")

print("User Information:", user_info)
print("Recent Activities:", recent_activities)
print("Followers:", followers)
```

Each Table with At Least 5 Columns

User Information Table: Includes columns such as age, country, and registration_date.

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Example 1: User Information Table (Using Hash)
def add_user(user_id, username, email, age, country, registration_date):
    user_key = f"user:{user_id}"
    redis_client.hset(user_key, "user_id", user_id)
    redis_client.hset(user_key, "username", username)
    redis_client.hset(user_key, "email", email)
    redis_client.hset(user_key, "age", age)
    redis_client.hset(user_key, "country", country)
    redis_client.hset(user_key, "registration_date", registration_date)

def get_user(user_id):
    user_key = f"user:{user_id}"
    user_info = redis_client.hgetall(user_key)
    return user_info
```

List of Recent Activities Table: Includes a timestamp along with each activity for better tracking.

```
# Example 2: List of Recent Activities Table (Using List)
def log_recent_activity(user_id, activity, timestamp):
    recent_activities_key = f"recent_activities:{user_id}"
    activity_data = f"{activity} ({timestamp})"
    redis_client.lpush(recent_activities_key, activity_data)
    redis_client.ltrim(recent_activities_key, 0, 4) # Keep only the latest 5 activities

def get_recent_activities(user_id):
    recent_activities_key = f"recent_activities:{user_id}"
    activities = redis_client.lrange(recent_activities_key, 0, -1)
    return activities
```

Set of Followers Table: Includes more details about each follower, such as follower_username, follower_email, and follower_since.

```
# Example 3: Set of Followers Table (Using Set)
def add_follower(user_id, follower_id, follower_username, follower_email, follower_since):
    followers_key = f"followers:{user_id}"
    follower_data = f"{follower_id}:{follower_username} ({follower_email}), {follower_since}"
    redis_client.sadd(followers_key, follower_data)

def get_followers(user_id):
    followers_key = f"followers:{user_id}"
    followers = redis_client.smembers(followers_key)
    return followers

# Example Usage
add_user("1", "john_doe", "john@example.com", "25", "USA", "2023-01-01")
log_recent_activity("1", "Logged in", "2023-01-10 10:00 AM")
add_follower("1", "2", "jane_doe", "jane@example.com", "2023-01-05")

user_info = get_user("1")
recent_activities = get_recent_activities("1")
followers = get_followers("1")

print("User Information:", user_info)
print("Recent Activities:", recent_activities)
print("Followers:", followers)
```

C. Inserting Data

Code Example for Inserting at Least 5 Records per Table

At least 5 users into the "User Information Table."

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Example 1: User Information Table (Using Hash)
def add_user(user_id, username, email, age, country, registration_date):
    user_key = f"user:{user_id}"
    redis_client.hset(user_key, "user_id", user_id)
    redis_client.hset(user_key, "username", username)
    redis_client.hset(user_key, "email", email)
    redis_client.hset(user_key, "age", age)
    redis_client.hset(user_key, "country", country)
    redis_client.hset(user_key, "registration_date", registration_date)

# Insert at least 5 records
for i in range(1, 6):
    add_user(str(i), f"user{i}", f"user{i}@example.com", str(20 + i), "Country" + str(i), f"2023-01-0{i}")
```

Recent activities for user 1 into the "List of Recent Activities Table."

```
# Example 2: List of Recent Activities Table (Using List)
def log_recent_activity(user_id, activity, timestamp):
    recent_activities_key = f"recent_activities:{user_id}"
    activity_data = f"{{activity}} ({{timestamp}})"
    redis_client.lpush(recent_activities_key, activity_data)
    redis_client.ltrim(recent_activities_key, 0, 4) # Keep only the latest 5 activities

# Insert recent activities for user 1
activities = ["Logged in", "Posted a message", "Updated profile", "Commented on a post", "Logged out"]
for activity in activities:
    log_recent_activity("1", activity, "2023-01-10 10:00 AM")
```

Followers for user 1 into the "Set of Followers Table."

```
# Example 3: Set of Followers Table (Using Set)
def add_follower(user_id, follower_id, follower_username, follower_email, follower_since):
    followers_key = f"followers:{user_id}"
    follower_data = f"{{follower_id}}:{{follower_username}} ({{follower_email}}), {{follower_since}}"
    redis_client.sadd(followers_key, follower_data)

# Insert followers for user 1
followers_data = [
    ("2", "jane_doe", "jane@example.com", "2023-01-05"),
    ("3", "bob_smith", "bob@example.com", "2023-01-07"),
    ("4", "alice_jones", "alice@example.com", "2023-01-09"),
    ("5", "charlie_brown", "charlie@example.com", "2023-01-11")
]

for follower_data in followers_data:
    add_follower("1", *follower_data)

# Display User Information
for i in range(1, 6):
    user_info = redis_client.hgetall(f"user:{i}")
    print(f"\nUser Information for user{i}: ", user_info)

# Display Recent Activities
recent_activities = get_recent_activities("1")
print("\nRecent Activities for user 1:", recent_activities)

# Display Followers
followers = get_followers("1")
print("\nFollowers for user 1:", followers)
```

D. Querying Data Examples

Code Examples for Various Query Operations

Example 1: Querying User Information Table

```
def get_users_by_age(age):  
    # Simulating a query for users with a specific age  
    matching_users = []  
    for user_id in range(1, 6):  
        user_info = redis_client.hgetall(f"user:{user_id}")  
        if user_info.get("age") == age:  
            matching_users.append(user_info)  
    return matching_users  
  
# Example: Get users with age 22  
users_with_age_22 = get_users_by_age("22")  
print("Users with age 22:", users_with_age_22)
```

Example 2: Querying List of Recent Activities Table

```
def filter_activities_by_keyword(user_id, keyword):  
    # Simulating a query for activities containing a keyword  
    recent_activities = get_recent_activities(user_id)  
    filtered_activities = [activity for activity in recent_activities if keyword.lower() in activity.lower()]  
    return filtered_activities  
  
# Example: Get recent activities for user 1 containing "Logged"  
filtered_activities = filter_activities_by_keyword("1", "Logged")  
print("Filtered Activities:", filtered_activities)  
  
Filtered Activities: ['Logged out (2023-01-10 10:00 AM)', 'Logged in (2023-01-10 10:00 AM)']
```

Example 3: Querying Set of Followers Table

```
def get_followers_with_email(user_id, email):  
    # Simulating a query for followers with a specific email  
    followers_data = get_followers(user_id)  
    matching_followers = [follower for follower in followers_data if email.lower() in follower.lower()]  
    return matching_followers  
  
# Example: Get followers for user 1 with email containing "jane"  
followers_with_email = get_followers_with_email("1", "jane")  
print("Followers with email containing 'jane':", followers_with_email)  
  
Followers with email containing 'jane': ['2:jane_doe (jane@example.com), 2023-01-05']
```

E. Saving and Ending Case

Properly Closing the Connection

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Perform Redis operations...

# Close the connection when done
redis_client.close()
```

Saving Changes

```
import redis

# Connect to the Redis server
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Perform Redis operations...

# Save data to disk synchronously (blocking)
redis_client.save()

# OR

# Save data to disk asynchronously (non-blocking)
redis_client.bgsave()
```

True

To manually trigger a save to disk, you can use the SAVE or BGSAVE command:

SAVE: Performs a synchronous save, blocking other clients while the save is in progress.

BGSAVE: Performs an asynchronous save in the background, allowing other operations to continue.

Conclusion

A variety of data structures, including strings, hashes, lists, and sets, are available in Redis, a flexible and fast in-memory data store. Because of its outstanding performance, which is mostly caused by its in-memory design, it is a great option for applications that require low-latency data access. Redis is primarily intended to be an in-memory database, even if it offers persistent techniques. Redis is commonly used for event-driven application development, with its publish/subscribe approach being leveraged for caching, real-time analytics, leaderboards, and messaging systems. Redis supports replication and clustering to manage big databases and high-throughput applications. It is renowned for its simplicity, usability, and scalability. Redis and MySQL are two distinct databases with different use cases and data structures.

Redis is an excellent choice for situations where quick access to data is essential since it is a key-value store that performs well in read-intensive tasks. Because it is a relational database, MySQL works well with applications that need structured data, transactions, and sophisticated queries. Although MySQL usually scales vertically or through sharding, Redis scales horizontally through replication and clustering. The application's particular requirements, such as the type of data, performance requirements, and intended data linkages, will determine which of Redis and MySQL is best.

References

- Fisher, T. (2020, December 31). *What is Ram and what does it do?*. Lifewire.
<https://www.lifewire.com/what-is-random-access-memory-ram-2618159>
- Python guide*. Redis. (n.d.). <https://redis.io/docs/connect/clients/python/>
- Kovačević, A. (2022, November 21). *How to install Redis on mac {with or without Homebrew}*. Knowledge Base by phoenixNAP.
<https://phoenixnap.com/kb/install-redis-on-mac>
- Python guide*. Redis. (n.d.). <https://redis.io/docs/connect/clients/python/>
- The advantages and disadvantages of working on Open source projects: Refine*. refine RSS. (2022a, April 29).
<https://refine.dev/blog/open-source-advantages-disadvantages/#learn-from-other-developers>
- What is Redis explained?*. IBM. (n.d.). <https://www.ibm.com/topics/redis>
- YouTube. (2023). *YouTube*. Retrieved November 30, 2023, from
<https://www.youtube.com/watch?v=IWJKRmFLn-g>.

Anson Ekau submission on GitHub:

ae kau21 / SQL-and-Relational-Databases-Final-Paper

Type to search

>

+

🕒

🔍

📧

👤

<>

Code

🕒

Issues

🔍

Pull requests

🕒

Actions

📁

Projects

🛡️

Security

📈

Insights

⚙️

Settings

Files

main

+

🔍

Go to file

🔍

.gitattributes

CS200 Final Paper_Redis.pdf

SQL-and-Relational-Databases-Final-Paper / CS200 Final Paper_Redis.pdf

...

ae kau21 Create CS200 Final Paper_Redis.pdf

10276e0 · now

History

3.32 MB

Code 55% faster with GitHub Copilot

📄

✎

⌵

CS200 FINAL: Installing and Using Redis to Create Databases

Anson Ekau, Mia Wong, Kandielynn Ueligitone, Christian Raquepo

School of Natural Sciences and Mathematics CS-200-01-1

Dr. Rylan Chong

December 1st, 2023