4장. 레디스 자료 구조 활용 사례

레디스의 자료 구조를 적절하게 활용해서 애플리케이션의 성능을 향상시키고 쉽게 개발할 수 있는 방법을 사례와 함께 알아봅시다.

왜 레디스 커맨드를 사용하나요?

🗣️ 애플리케이션 레벨에서 구현하면 되는 거 아냐?

- 애플리케이션에서 구현 시 메모리 영역으로 데이터를 가져간 후 가공해야 한다.
- 레디스 내장 함수를 사용하면 매우 짧은 시간 안에 엄청난 양의 작업을 처리할 수 있다.

1️⃣ 실시간 리더보드 with `sorted set`

절대적 리더보드 vs 상대적 리더보드

절대적 리더보드

- 모든 유저를 정렬시켜 상위권의 목록만 표시
- 모든 사용자에게 같은 데이터를 보여줌

상대적 리더보드

- 사용자가 속한 그룹 내 상대적인 순위를 표시
- 사용자마다 다른 데이터를 보여줌

`sorted set`

데이터는 **저장될 때부터 정렬**되어 들어간다.

- 기본적으로 `set`이기 때문에 데이터는 중복 저장되지 않는다.
- 아이템을 저장할 때 이미 존재하면 기존 데이터의 스코어만 새로 업데이트한다.
- 스코어가 업데이트되면 데이터의 순서도 재정렬된다.
- 스코어가 같으면 사전 순으로 정렬

명령어

```
# 데이터 저장
> ZADD daily-score:240320 28 player:286
(integer) 1

# 데이터 확인(오름차순 정렬)
> ZRANGE daily-score:240320 0 -1 withscores
1) "player:286"
2) "28"
3) "player:101"
4) "45"
...
9) "player:234"
10) "400"

# 데이터 확인(내림차순 정렬)
# 0번 ~ 2번 인덱스 (상위 3개)
> ZREVRANGE daily-score:240320 0 2 withscores
1) "player:234"
2) "400"
3) "player:24"
4) "357"
5) "player:143"
6) "199"

# 원자적으로 스코어 증가
# 스코어 100 증가 후 재정렬
> ZINCRBY daily-score:240320 100 player:286

# 지정한 키에 연결된 각 아이템의 스코어 합산
# ZUNIONSTORE 합산키이름 합산키개수 합산키 ...
> ZUNIONSTORE weekly:2403-3 3 daily:240318 daily:240319 daily

# 가중치를 두어 스코어 합산
# ZUNIONSTORE 합산키이름 합산키개수 합산키 ... weights 가중치 ...
> ZUNIONSTORE weekly:2403-3 3 daily:240318 daily:240319 daily
```

2️⃣ 최근 검색 기록 with `sorted set`

저장할 때 중복을 허용하지 않기 때문에 **스코어로 시간을 사용**하면 최근 검색 기록을 자동 정렬할 수 있다.

같은 키워드를 다시 검색하면 아이템의 스코어가 현재 시간으로 업데이트 되고 저장 순서도 변경된다.

음수 인덱스

음수 인덱스를 사용해서 데이터를 삭제하면 최근 데이터 N개 유지 등의 작업을 수월하게 할 수 있다.

음수 인덱스는 -1부터 시작한다. (제일 마지막 값 == -1)

N개 이상의 데이터가 저장되지 않게 강제하기

```
> ZADD search-keyword 20240320153043 20 황금비
1

> ZREMRANGEBYRANK search-kewyrod -6 -6
# 음수 인덱스 -6번째 데이터 삭제
# 6개 데이터가 있을 경우 -6번 음수 인덱스는 일반 인덱스 0 -> 데이터 5개
# 데이터가 5개 이하이면 -6번 인덱스는 존재하지 않기 때문에 데이터가 삭제되
```

3️⃣ 태그 기능 with `sorted set`

```
# 47번 게시물에 태그 3개 추가(IT, REDIS, DataStore)
> SADD post:47:tags IT REDIS DataStore
3
# 22번 게시물에 태그 2개 추가(IT, python)
> SADD post:22:tags IT python
2
```

특정 태그를 포함한 게시물 확인하기

데이터 저장 시 게시물 기준의 set, 태그 기준의 set에 각각 데이터를 넣어주면 쉽게 구현할 수 있다.

```
# (게시물 기준) 53번 게시물에 태그 3개 추가(DataStore, IT, MySQL)
> SADD post:53:tags DataStore IT MySQL
3
# 태그 3개에 게시물 번호(53) 추가
> SADD tag:DataStore:posts 53
1
> SADD tag:IT:posts 53
1
> SADD tag:MySQL:posts 53
1

# 특정 태그를 갖고 있는 게시물 조회
> SMEMBERS tag:IT:posts
1) "22"
2) "47"
3) "53"

# 여러 태그를 모두 가지고 있는 게시물 조회(set의 교집합)
> SINTER tag:IT:posts tag:DataStore:posts
1) "47"
2) "53"
```

4️⃣ 랜덤 데이터 추출

레디스를 사용하면 O(1)의 시간 복잡도를 이용해 랜덤 데이터를 추출할 수 있다.

- **RANDOMKEY**: 레디스에 저장된 전체 키 중 하나를 무작위로 반환
- **HRANDFIELD**: hash에 저장된 아이템 중 하나를 무작위로 반환
- **SRANDMEMBER**: set에 저장된 아이템 중 하나를 무작위로 반환
- **ZRANDMEMBER**: sorted set에 저장된 아이템 중 하나를 무작위로 반환

```
# hash에 저장된 랜덤 유저 반환
> HRANDFIELD user:hash
"Id:4615"
# hash에 저장된 랜덤 유저 값과 함께 반환
# HRANDFIELD 키이름 COUNT옵션(n개) WITHVALUES(값과 함께 반환)
```

```
> HRANDFIELD user:hash 1 WITHVALUES
1) "Id:4615"
2) "geumbi"
```

- `COUNT` 옵션은 양수로 설정 시 중복되지 않는 랜덤 데이터가 반환되고 음수로 설정하면 데이터가 중복해서 반환될 수 있다.
- `SRANDMEMBER`, `ZRANDMEMBER`에서 `WITHSCORE` 옵션을 사용하면 필드에 연결된 값도 함께 반환한다.

5️⃣ 카운팅

좋아요 처리하기

1. 댓글 id를 기준으로 set을 생성한다.
2. 좋아요를 누른 유저의 id를 set에 저장한다.
3. 댓글별로 좋아요를 누른 수를 확인한다.

```
# 좋아요한 유저 id 추가
> SADD comment-like:12254 957 234 125
(integer) 3
# 좋아요 수 확인
> SCARD comment-like:12254
(integer) 3
```

읽지 않은 메시지 수 확인하기

사용자의 ID를 키, 채널의 ID를 아이템의 키로 활용해서 숫자 형태의 메시지 카운트를 관리한다.

```
# ID가 234인 사용자가 3456 채널에서 새로운 메시지를 수신(+1)한 경우
> HINCRBY user:234 channel:3456 1
(integer) 1
# ID가 123인 사용자가 3135 채널에서 메시지를 삭제(-1)한 경우
> HINCRBY user:123 channel:3135 -1
(integer) 26
```

DAU 구하기

실제 서비스를 이용한 **사용자의 유니크한 수**를 파악할 수 있다.

레디스의 비트맵을 이용하면 하루 1000만 명 이상의 유저가 방문하는 큰 서비스의 DAU도 메모리 측면에서 효율적으로 관리할 수 있다.

레디스의 비트맵

- 별도의 자료구조는 아니다.
- string 자료 구조에 bit 연산을 할 수 있도록 구현됐다.
- 사용자 id가 0 이상의 정수인 경우 string 자료 구조에서 하나의 비트로 표현될 수 있으므로 이를 활용한다.

```
# ID가 14인 사용자가 방문
> SETBIT uv:20240320 14 1
(integer) 0
# 해당 일자에 접근한 사용자 수 확인
> BITCOUNT uv:20240320
(integer) 3
# AND 연산을 이용한 3일 연속 방문한 사용자 수 확인
> BITOP AND event:202403 uv:20240318 uv:20240319 uv:20240320
(integer) 2
# 값 조회
> GET event:202403
"\x01\x02" # string 자료 구조로 저장되므로 문자 단위로 비트 연산을 수행
```

6️⃣ 애플리케이션 미터링 with `hyperloglog`

hyperloglog

- 집합 내의 유일한 데이터의 개수를 카운팅해야 하는 경우
- 1% 미만의 오차가 허용되는 경우
- 카운팅할 때 사용한 정확한 데이터를 다시 확인하지 않아도 되는 경우

→ 위 조건을 만족하는 경우 `hyperloglog`를 사용하면 최소한의 메모리를 사용해 중복되지 않는 데이터 수를 계산할 수 있다.

set과 비슷하지만 저장되는 용량이 12KB로 고정되기 때문에 공간을 효율적으로 사용할 수 있다.

7️⃣ 위치 기반 애플리케이션 with Geospatial Index

geo set

- 위치 공간 관리에 특화된 자료구조
- 경도, 위도의 쌍으로 저장
- 내부적으로 **sorted set** 구조로 저장

```
# 위치 저장
GEOADD user 50.234235234235 14.232342352341 142
GEOADD restaurant 50.34523436 14.36243525423 outback
# 위치 조회
> GEOPOS restaurant outback
1) 1) 50.34523436
    2) 14.36243525423

# 특정 위치 근처(반경 1km)의 식당 조회
> GEOSEARCH restaurant fromlonlat 50.234235234235 14.23234235
1) "outback"
```

- **fromlonlat**: 직접 경도, 위도를 지정하여 검색
- **frommember**: 동일한 데이터 셋 내에서 검색하는 경우 위도, 경도 없이 원하는 데이터 검색 가능
- **byradius**: 반지름 값을 기준으로 검색
- **bybox**: width, height 값을 추가 지정하여 특정 위치를 중심으로 한 직사각형 영역 기준으로 검색