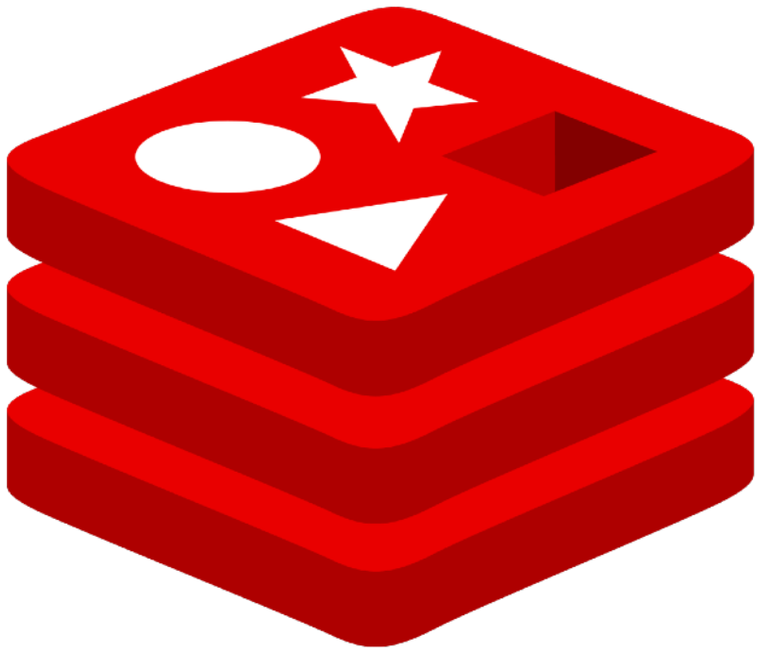


Banco de dados NoSQL - Chave/Valor com Redis

Sets, Sorted Set and Transactions

Prof. Gustavo Leitão

Sets

(Conjuntos)

SETS

Sets é um tipo especial de lista que não permite dados repetidos.

```
$ SADD my-set “Hello"

$ SADD my-set “Hello"

$ SADD my-set “World”

$ SMEMBERS my-set
```

```
redis> SADD myset "Hello"
(integer) 1
redis> SADD myset "World"
(integer) 1
redis> SMEMBERS myset
1) "World"
2) "Hello"
redis>
```

SETS - Comandos importantes

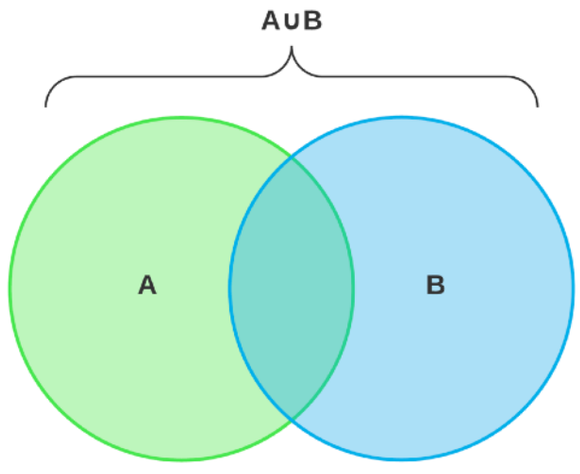
`SCARD my-set` — Retorna o número de elementos do conjunto (Cardinalidade)

`SINTER my-set1 my-set2` — Retorna interceção entre os conjuntos (dois ou mais)

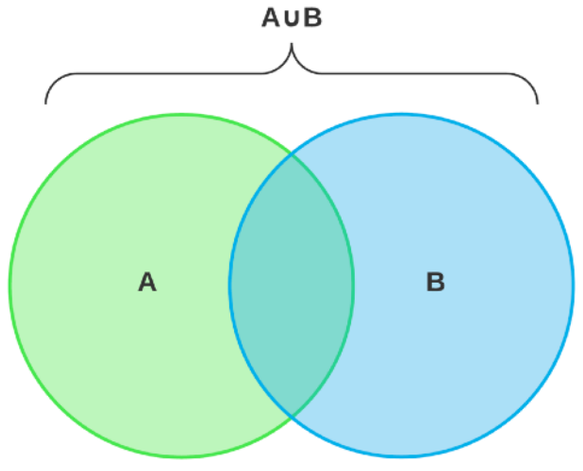
`SINTERSTORE dest my-set1 my-set2` — Armazena em "dest" a interceção entre os conjuntos (dois ou mais)

`SUNION my-set1 my-set2` — Retorna a união entre os conjuntos (dois ou mais)

`SUNIONSTORE dest my-set1 my-set2` — Armazena em "dest" a união entre os conjuntos (dois ou mais)



SETS - Comandos importantes



`SDIFF my-set1 my-set-2` — Retorna a diferença entre o primeiro conjunto e os demais, ou seja: (my-set-1 - my-set-2)

`SDIFFSTORE dest my-set1 my-set-2` — Armazena a diferença entre o primeiro conjunto e os demais, ou seja: (my-set-1 - my-set-2)

`SPOP my-set` — Remove um elemento aleatório do conjunto

`SREM my-set` "member" — Remove o elemento "member" do conjunto

Sorteded Sets

(Conjuntos Ordenados)

Sorted Set

Conjunto onde os elementos são mantidos seguindo determinada ordem.

```
ZADD key [NX|XX] [CH] [INCR] score member
[score member ...]
```

Opções

XX: Atualiza apenas elementos que já existem. Nunca adiciona novos elementos.
NX: Não atualiza elementos existentes. Sempre adiciona novos elementos.

CH: Modifica o retorno para informar todos registros que sofreram alguma modificação. Normalmente o retorno retorna apenas o número de elementos adicionados.

INCR: Quando esta opção é especificada, o score do elemento é adicionado ao valor passado.

Sorted Set

Alguns comandos importantes

```
$ ZRANGE —> Retorna os elementos em um dado range (menor > maior)
```

```
$ ZCARD —> Retorna a cardinalidade (número de elementos)
```

```
$ ZCOUNT —> Retorna o numero de elementos com score em um
dado intervalo.
```

```
$ ZSCORE —> Retorna o score de um dado elemento
```

```
$ ZRANK —> Retorna a posição de um dado elemento
```

Sorted Set

Alguns comandos importantes

```
$ ZPOPMIN —> Remove e retorna elemento com menor score
```

```
$ ZPOPMAX —> Remove e retorna elemento com maior score.
```

- BZPOPMAX
- BZPOPMIN
- ZADD
- ZCARD
- ZCOUNT
- ZINCRBY
- ZINTERSTORE
- ZLEXCOUNT
- ZPOPMAX
- ZPOPMIN
- ZRANGE
- ZRANGEBYLEX
- ZRANGEBYSCORE
- **ZRANK**
- ZREM
- ZREMRANGEBYLEX
- ZREMRANGEBYRANK
- ZREMRANGEBYSCORE
- ZREVRANGE
- ZREVRANGEBYLEX
- ZREVRANGEBYSCORE
- ZREVRANK
- ZSCAN
- ZSCORE
- ZUNIONSTORE

Exercício:

Utilizando Sorted Set implemente um sistema de ranqueamento de um jogo. As seguintes operações devem estar disponíveis:

1- Adicionar N pontos ao jogador X
2- Remover N Pontos ao jogador X
3- Lista dos TOP 10
4- Jogador na Enésima posição no rank

De preferência implemente rotas em um servidor web para cada operação.
Extra: Use JSON para retorno dos resultados

Transactions

Transações com Redis

É possível criar uma transação no Redis para garantir que todos os comandos sejam executados atomicamente.

MULTI — Inicia uma Transação
EXEC — Executa a transação
DISCARD — Aborta a transação

```
$ MULTI
$ INCR foo
$ INCR bar
$ EXEC
```

```
> MULTI
OK
> INCR foo
QUEUED
> INCR bar
QUEUED
> EXEC
1) (integer) 1
2) (integer) 1
```

Transações com Redis

Redis não suporta roll backs!

Algumas justificativas para isso:

1. Os comandos só falham se houver erro de sintaxe ou tipos de dados errados
2. Um das premissas do Redis é ser rápido e simples e roll back impactaria nessas premissas.

Transações com Redis

Redis implementa *locks* otimistas.

Como garantir que a transação não seja executada, caso haja alteração por outro cliente em alguma chave que será impactada?

Em vez de bloquear as alterações em uma chave durante uma transação o Redis implementa algo mais leve e otimista através do comando WATCH.

No exemplo abaixo, se a chave “foo" for alterada por outro cliente após o comando WATCH, a transação não será executada! Isso que chama-se lock otimista.

```
$ WATCH foo
$ MULTI
$ INCR foo
$ EXEC
```

```
WATCH mykey
val = GET mykey
val = val + 1
MULTI
SET mykey $val
EXEC
```

Bulk insert

Inserção em Lote

Talvez você precise inserir um grande conjunto de dados para o Redis.

Inserir um grande conjunto 1 a 1 é bem ineficiente devido ao *roud trip* de cada comando.

Então, como fazer??

Inserção em Lote

Passo 1 - Crie um arquivo com os comandos a serem executados

```
SET Key0 Value0
SET Key1 Value1
...
SET KeyN ValueN
```

Passo 2 - Execute o seguinte comando

```
cat data.txt | redis-cli --pipe
```

Inserção em Lote

Exemplo

```
$ docker exec -it #id bash
$ touch data.txt
$ vi data.txt
$ cat data.txt | redis-cli —pipe
```

Obrigado!

Prof. Gustavo Leitão