Disciplina:

Banco de Dados NoSQL

Professor: Anderson Theobaldo

PUC Minas

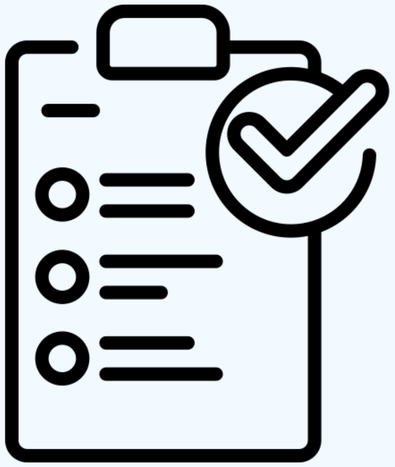
Aula 2:

Banco de Dados NoSQL Tipo Chave/Valor

PUC Minas

Nesta Aula

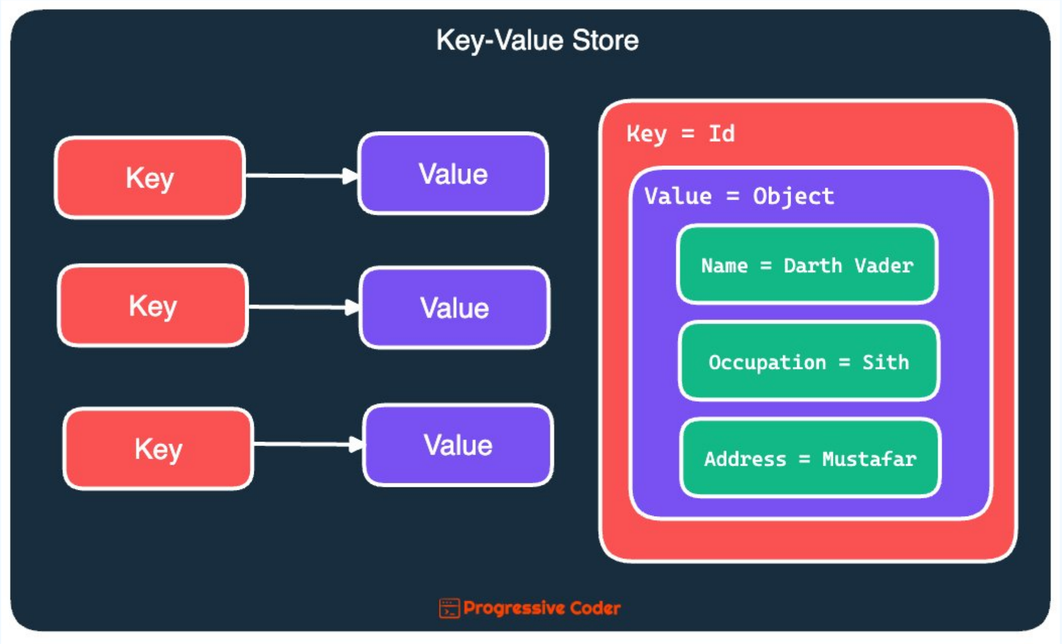
- Definição
- Características
- Principais opções do mercado
- Banco de Dados Redis
- Demonstração de uso



PUC Minas

Definição

- Bancos de dados NoSQL Chave/Valor são sistemas de armazenamento de dados que organizam informações na forma de pares simples de chave e valor.
- Em contraste com bancos de dados relacionais, que seguem um modelo tabular com esquemas rígidos, os bancos de dados NoSQL chave/valor oferecem uma abordagem mais flexível e simplificada para armazenar dados.



PUC Minas

Características Principais

Simplicidade

São simples de entender e usar, pois armazenam dados como pares chave/valor, eliminando a necessidade de um esquema complexo.

Escalabilidade Horizontal

São altamente escaláveis horizontalmente, o que significa que podem lidar com grandes volumes de dados distribuídos em vários servidores.

Alta performance

são otimizados para operações de leitura e gravação de pares chave/valor, oferecendo alta performance para cenários de acesso intensivo a dados.

Flexibilidade no esquema

Não têm um esquema fixo, permitindo que diferentes chaves tenham valores de diferentes tipos e estruturas.

Baixa consistência

Frequentemente priorizam disponibilidade e tolerância a partições em detrimento da consistência forte, podendo levar a resultados inconsistentes em falhas de rede ou partições, embora normalmente garantam consistência eventual.

Suporte a dados semiestruturados

Capacidade de lidar com dados semiestruturados ou não estruturados, tornando-os ideais para cenários onde a estrutura dos dados pode variar ou evoluir ao longo do tempo.

Ideal para cenários específicos

São comumente usados em cenários onde a estrutura dos dados é simples e previsível, como armazenamento de sessões de usuário, caching, gerenciamento de configurações e armazenamento de metadados.

Aplicações Comuns

Caching

- São frequentemente utilizados para implementar sistemas de cache em aplicativos web e móveis.
- Armazenar dados em cache ajuda a reduzir o tempo de resposta e aliviar a carga dos servidores, melhorando a experiência do usuário.

Armazenamento de Sessões

- Em muitas aplicações web, é necessário armazenar informações de sessão do usuário, como autenticação e preferências.
- Os bancos de dados chave/valor oferecem uma maneira eficiente de armazenar e recuperar esses dados de forma rápida e escalável.

Aplicações Comuns

Gerenciamento de Configurações

- Configurações de aplicativos, como parâmetros de ambiente e configurações de sistema, podem ser armazenadas em bancos de dados NoSQL chave/valor.
- Esses bancos de dados são adequados para essa finalidade devido à sua simplicidade e facilidade de escalabilidade.

Filas de Mensagens e Pub/Sub

- Os sistemas de mensagens e publicação/assinatura (pub/sub) podem utilizar bancos de dados chave/valor para armazenar mensagens e notificações temporárias.
- Esses bancos de dados oferecem uma estrutura simples e eficiente para gerenciar a entrega de mensagens entre os componentes do sistema.

PUC Minas

Raking Dos Mais Utilizados

☐ include secondary database models — 70 systems in ranking, April 2024

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Redis	Key-value, Multi-model	156.44	-0.56	-17.11
2.	2.	2.	Amazon DynamoDB	Multi-model	77.57	-0.15	+0.12
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	29.85	-0.54	-5.23
4.	4.	4.	Memcached	Key-value	20.74	-0.07	-1.25
5.	↑ 6.	5.	etcd	Key-value	7.64	-0.02	-0.99
6.	↓ 5.	6.	Hazelcast	Key-value, Multi-model	6.87	-0.79	-0.89
7.	7.	7.	Aerospike	Multi-model	6.10	-0.41	-0.30
8.	8.	8.	Ehcache	Key-value	5.23	-0.32	-0.91
9.	9.	↑ 10.	Riak KV	Key-value	4.44	-0.51	-0.75
10.	↑ 14.	↑ 20.	InterSystems IRIS	Multi-model	4.39	+0.57	+1.19

Fonte: DB-ENGINES

PUC Minas

Principais Operações

Inserção (PUT)

Adicionar um novo par chave/valor ao banco de dados.

Recuperação (GET)

Obter o valor associado a uma chave específica.

Atualização (PUT ou UPDATE)

Modificar o valor associado a uma chave existente.

Exclusão (DELETE)

Remover um par chave/valor do banco de dados.

Escaneamento (SCAN)

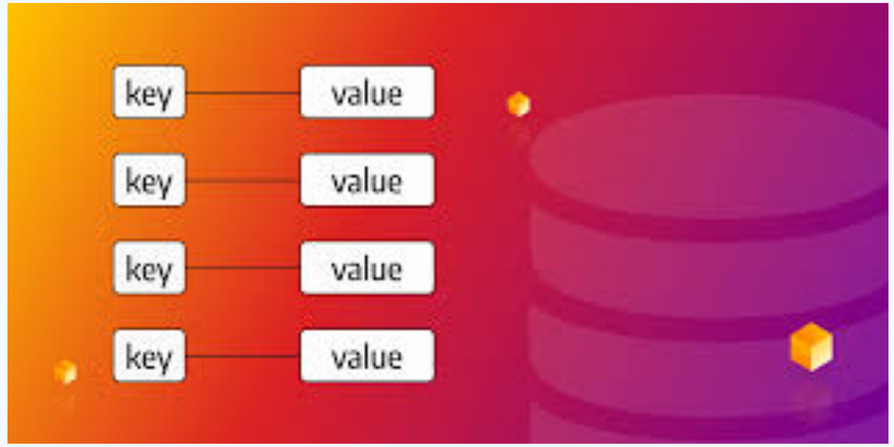
Percorrer todas as chaves ou uma faixa de chaves no banco de dados.

Expiração (EXPIRE)

Definir um tempo de vida para um determinado par chave/valor, após o qual ele é automaticamente removido do banco de dados.

Operações atômicas

Alguns bancos de dados NoSQL chave/valor oferecem operações atômicas, como incremento (INCR) e decremento (DECR), que garantem que várias operações ocorram como uma única transação para manter a consistência dos dados.




redis

O Que é Redis

- Redis é um acrônimo de REmote DIctionary Server.
- É um software open source de armazenamento de dados chave-valor em memória.
- Oferece um conjunto de estruturas versáteis que permite a fácil criação de várias aplicações personalizadas.
- É considerado a melhor opção devido ao seu alto desempenho por conta do armazenamento dos dados serem em memória ao invés do tradicional em discos.
- É um dos bancos de chave-valor mais conhecido atualmente.

PUC Minas

Benefícios

Desempenho muito rápido

Armazena todos os dados na memória principal do servidor, ao contrário de outros sistemas que usam disco ou SSDs. Isso elimina atrasos de busca e permite operações rápidas, geralmente executadas em menos de 10 milissegundos.

Replicação e persistência

Utiliza uma arquitetura mestre/subordinado e suporta replicação assíncrona para distribuir dados entre vários servidores subordinados. Isso resulta em melhor desempenho de leitura e recuperação em caso de falhas do servidor principal.

Estruturas de dados na memória

Suporta armazenamento de chaves mapeadas para vários tipos de dados, incluindo strings, listas, conjuntos, hashes e HyperLogLogs, permitindo a manipulação de diferentes estruturas de dados na memória.

Versatilidade e facilidade de uso

Oferece ferramentas como PUB/SUB para mensagens em tempo real e chaves com TTL para autoexclusão, facilitando o desenvolvimento e operações ao evitar sobrecarga com itens desnecessários.

Casos de Uso

Aprendizado de Máquina e IA

Em projetos de aprendizado de máquina e inteligência artificial, o Redis é utilizado para armazenar e processar modelos de machine learning, caching de resultados de inferência, gerenciamento de filas de tarefas de treinamento e colaboração de dados entre equipes de cientistas de

Análise de Dados

Em plataformas de análise de dados, o Redis é utilizado para caching de resultados de consultas, armazenamento de dados de séries temporais, indexação e consulta de dados geoespaciais, e processamento de dados em tempo real para geração de insights.

Loja Virtual

É utilizado para caching de páginas, armazenamento de carrinhos de compras, gerenciamento de sessões de usuários, recomendação de produtos personalizados e processamento de pedidos em tempo real.

Classificações em tempo real

É usado para gerenciamento de sessões de jogadores, armazenamento de dados de perfil e progresso do jogador, armazenamento de rankings e líderes de placar, e implementação de chats e notificações em tempo real.

Instalação

Linux

https://redis.io/docs/getting-started/installation/install-redis-on-linux/

macOS

https://redis.io/docs/getting-started/installation/install-redis-on-mac-os/

Windows

https://redis.io/docs/getting-started/installation/install-redis-on-windows/

Simulador

https://try.redis.io/

Explorando Redis com o CLI

- O Redis fornece um utilitário de linha de comando(redis-cli) que pode ser usado para enviar comandos.
- Executar redis-cli seguido por um nome de comando e seus argumentos enviará este comando para a instância do Redis em execução.
- A primeira coisa a fazer para verificar se o Redis está funcionando corretamente é enviar um comando PING usando redis-cli:

```
> redis-cli Ping
PONG
```

Explorando Redis com o CLI

- Uma das maneiras de executar redis-cli é sem argumentos
- O programa iniciará no modo interativo.
- Você pode digitar comandos diferentes e ver suas resposta

```
$ redis-cli
redis 127.0.0.1:6379> ping
PONG

redis 127.0.0.1:6379> set chave valor
OK

redis 127.0.0.1:6379> get chave
"valor"
```

PUC Minas

Obtendo informações do servidor Redis

- O comando INFO é usado para obter informações sobre o servidor Redis.

```
127.0.0.1:6379> info
# Server
redis_version:7.2.4
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:2a5e82a4865cf90d
redis_mode:standalone
os:Linux 6.6.22-linuxkit x86_64
arch_bits:64
monotonic_clock:POSIX clock_gettime
multiplexing_api:epoll
atomicvar_api:c11-builtin
gcc_version:12.2.0
process_id:1
process_supervised:no
run_id:70d09c26f865b021428805c995d265cdc5abaab5
tcp_port:6379
```

```
127.0.0.1:6379> info cpu
# CPU
used_cpu_sys:5.113582
used_cpu_user:2.718870
used_cpu_sys_children:0.001277
used_cpu_user_children:0.004928
used_cpu_sys_main_thread:5.094355
used_cpu_user_main_thread:2.708874
127.0.0.1:6379>
```

Selecionando o banco de dados

- O comando SELECT no Redis é usado para selecionar o banco de dados com o qual desejamos interagir.
- O Redis suporta múltiplos bancos de dados (também conhecidos como "índices de banco de dados") numerados de 0 a 15 por padrão.
- Quando conectamos a um servidor Redis, por padrão, estaremos conectado ao banco de dados 0.

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379>
```

Chaves

- As chaves do Redis são binárias seguras, isso significa que você pode usar qualquer sequência binária como chave, desde uma string como “cidade" até o conteúdo de um arquivo JPEG.
- A string vazia também é uma chave válida.
- Chaves longas não é uma boa ideia. Não apenas em termos de memória, mas devido a comparações de chave dispendiosas.
- O tamanho máximo de chave permitido é 512 MB.

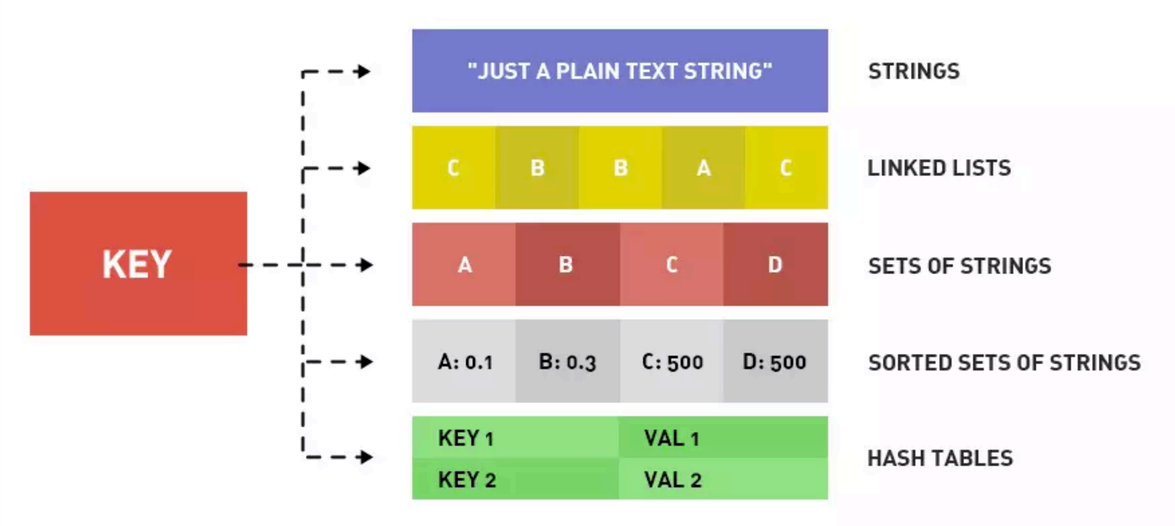
Convenção Para Nomear Chaves

- Utilizar : para compor um “namespace” na chave é uma convenção muito utilizada no Redis.
- O formato de chave muito comum assemelha-se com tipo-de-objeto:identificador.
- Exemplo:
  - usuario:1200
  - Sendo que usuario é o tipo de objeto, e o valor 1200 representa o id do usuário.

PUC Minas

Tipos de Dados

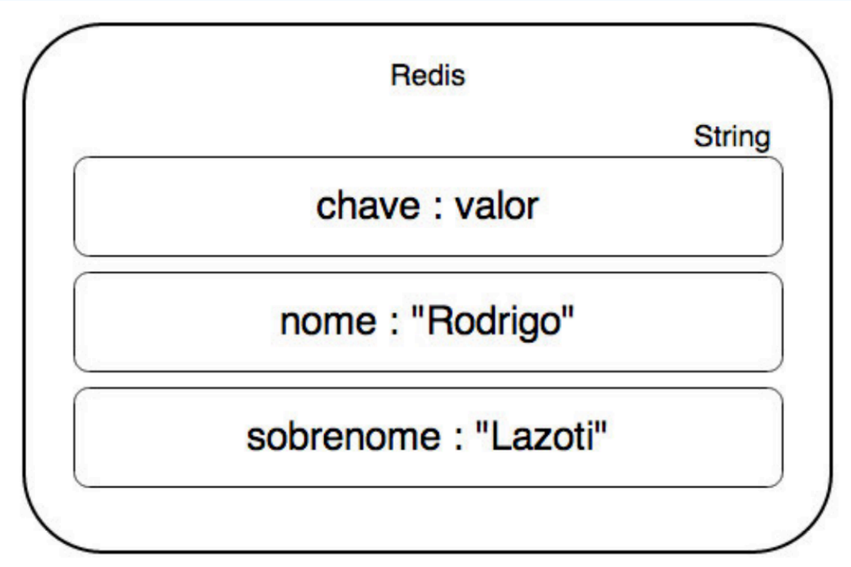
- Strings: tipo de dados mais básico que representa uma sequência de bytes.
- Lists: são listas de strings classificadas por ordem de inserção
- Sets: são coleções não ordenadas de strings exclusivas.
- Hashes: são tipos de registro modelados como coleções de pares de chave/valor.
- Sorted Sets: são coleções de strings exclusivas que mantêm a ordem pela pontuação associada a cada string.



PUC Minas

Tipo de Dado - String

- String é o tipo de dado mais comum disponível no Redis.
- Este tipo de dado é muito similar ao tipo string que vemos em outras linguagens de programação, como Java ou Ruby.
- No Redis, um valor do tipo string pode conter um tamanho de no máximo 512 Megabytes.



Principais Comandos

- **set** chave valor — armazena a chave e seu respectivo valor. Caso já exista uma chave definida, seu valor é sobrescrito;
- **get** chave — retorna o valor correspondente à chave informada;
- **mget** chave [chave ...] — retorna os valores correspondentes às chaves informadas;
- **mset** chave valor [chave valor ...] — armazena um ou mais conjuntos de chave valor. Caso uma chave informada já exista, seu valor será sobrescrito pelo novo;
- **append** chave valor — adiciona o valor a uma chave existente, ou cria uma nova chave (caso esta ainda não exista) com seu respectivo valor;
- **del** chave [chave ...] — remove a(s) chave(s) informada(s) e seu(s) respectivo(s) valor(es);
- **getrange** chave inicio fim — retorna uma parte da string armazenada conforme a chave informada;
- **Strlen** chave — retorna o tamanho da string armazenada conforme a chave informada.

Exemplos

- Instrução SET armazena a chave e seu respectivo valor. Caso já exista uma chave definida, seu valor é sobrescrito.
- Instrução GET retorna o valor correspondente à chave informada.

```
> set curso "Engenharia de Dados"
"OK"

> get curso
"Engenharia de Dados”

> set 10350 "José Ferreira"
“OK"

> get 10350
"José Ferreira"
```

Exemplos

- A instrução SET possui os seguintes parâmetros para configuração da expiração da chave:
  - EX expiração em segundos
  - PX expiração em milissegundos
  - NX indica que irá inserir a chave somente se não existir antes
  - XX indica que irá inserir a chave somente se já existir antes

```
> set uf “Minas Gerais” EX 5 NX
“OK”

> set uf “Rio de Janeiro" PX 600 NX
(nil)

> set uf “Rio de Janeiro" PX 600 XX
“OK”
```

Exemplos

- As seguintes instruções podem ser utilizadas para retornar tempo de expiração de uma chave:
- TTL retorna o tempo para expiração em segundos.
- PTTL retorna o tempo para expiração em milissegundos

```
> set uf “Minas Gerais” EX 5 NX
“OK”

> tll uf
(integer) 53

> pttl uf
(integer) 2400
```

Exemplos

- Instrução PERSIST remove o tempo de expiração de uma chave.

```
> set uf "Minas Gerais" EX 5 NX
"OK"

> tll uf
(integer) 53

> persist uf
(integer) 1

> tll uf
(integer) -1
```

Exemplos

- Instrução MSET armazena um ou mais conjuntos de chave/valor. Caso uma chave informada exista, seu valor será sobrescrito pelo novo.
- Instrução MGET retorno os valores correspondentes às chaves informadas.

```
> mset disciplina Matemática 3 Masculino uf “Minas Gerais"
“OK"

> mget disciplina 3 uf
1)  "Matemática"
2)  "Masculino"
3)  "Minas Gerais"
```

Exemplos

- Instrução APPEND adiciona o valor a uma chave existente, ou cria uma nova chave (caso esta ainda não exista) com seu respectivo valor

```
> set curso Engenharia
"OK"

> append curso "de Dados"
18

> get curso
“Engenharia De Dados”
```

Exemplos

- Instrução DEL remove a(s) chave(s) informada(s) e seu(s) respectivo(s) valor(es).
- Instrução EXISTS verifica a existência da chave informada.

```
> mset disciplina Matemática 3 Masculino uf “Minas Gerais"
“OK”

> del uf
(integer) 1

> get uf
(nil)

> exists uf
(integer) 0
```

Exemplos

- Instrução GETRANGE retorna uma parte da string armazenada conforme a chave informada.
- Instrução STRLEN retorna o tamanho da string armazenada conforme a chave informada.

```
> mset disciplina Matemática 3 Masculino uf “Minas Gerais"
“OK”

> getrange uf 0 4
“Minas”

> strlen uf
(Integer) 12
```

Tipo de Dado - List

- Dados do tipo list (ou lista) são basicamente listas de strings ordenadas pela ordem de inserção de cada item.
- Frequentemente usado para implementação de pilhas e filas.
- O tamanho máximo de elementos contidos em uma única lista é de mais de quatro bilhões de elementos ou, sendo mais preciso, 4294967295 elementos.

Principais Comandos

- LPUSH adiciona um ou mais valores ao topo (head) da lista definida pela chave.
- RPUSH adiciona um ou mais valores ao final (tail) da lista definida pela chave.
- LPOP remove e retorna um elemento do inicio de uma lista.
- RPOP remove e retorna um elemento do final da lista.
- LLEN retorna a quantidade de elementos da lista.
- LRANGE retorna um range de elementos de uma lista.
- LTRIM reduz uma lista ao intervalo especificado de elementos.
- LMOVE move atomicamente elementos de uma lista para outra.
- LINSERT Insere um elemento na lista, armazenada em uma chave, antes ou depois do valor de referência.
- LSET altera o valor de um posição(índice) específico

Exemplos

- Trate uma lista como uma fila (primeiro a entrar, primeiro a sair):

```
> lpush pedido:fila:ids 101 102 103 104
(integer) 1

> rpop pedido:fila:ids
"101"

> rpop pedido:fila:ids
“102”

> rpop pedido:fila:ids
"103"

> rpop pedido:fila:ids
“104”
```

PUC Minas

Exemplos

- Trate uma lista como uma pilha (último a entrar, primeiro a sair):

```
> lpush pedido:fila:ids 101 102 103 104
(integer) 1

> lpop pedido:fila:ids
"104"

> lpop pedido:fila:ids
“103”

> lpop pedido:fila:ids
"102"

> lpop pedido:fila:ids
“101”
```

Exemplos

- Verifique o comprimento da lista:

```
> llen pedido:fila:ids
(integer) 0
```

Exemplos

- Desloque atomicamente um elemento de uma lista e empurre para outra:

```
[127.0.0.1:6379> lpush pedido:fila1:ids 101 102
(integer) 2
[127.0.0.1:6379> lpush pedido:fila2:ids 103 104
(integer) 2
[127.0.0.1:6379> lmove pedido:fila1:ids pedido:fila2:ids left right
"102"
[127.0.0.1:6379> lrange pedido:fila1:ids 0 -1
1) "101"
[127.0.0.1:6379> lrange pedido:fila2:ids 0 -1
1) "104"
2) "103"
3) "102"
127.0.0.1:6379>
```

PUC Minas

Exemplos

- Para criar uma lista limitada que nunca ultrapasse 5 elementos, você pode chamar LTRIM após cada chamada para LPUSH:

```
[127.0.0.1:6379> lpush notifications:user:1 "voce recebeu um email."
(integer) 1
[127.0.0.1:6379> lpush notifications:user:1 "voce recebeu um email."
(integer) 2
[127.0.0.1:6379> lpush notifications:user:1 "voce recebeu um email."
(integer) 3
[127.0.0.1:6379> lpush notifications:user:1 "voce recebeu um email."
(integer) 4
[127.0.0.1:6379> lpush notifications:user:1 "voce recebeu um email."
(integer) 5
[127.0.0.1:6379> lrange notifications:user:1 0 -1
1) "voce recebeu um email."
2) "voce recebeu um email."
3) "voce recebeu um email."
4) "voce recebeu um email."
5) "voce recebeu um email."
[127.0.0.1:6379> ltrim notifications:user:1 0 3
OK
[127.0.0.1:6379> lrange notifications:user:1 0 -1
1) "voce recebeu um email."
2) "voce recebeu um email."
3) "voce recebeu um email."
4) "voce recebeu um email."
127.0.0.1:6379>
```

Exemplos

- Para criar uma lista limitada que nunca ultrapasse 100 elementos, você pode chamar LTRIM após cada chamada para LPUSH:

```
> lpush lista "Verde"
(integer) 1

> lpush lista "Amarelo"
(integer) 2

> linsert lista BEFORE "Branco" "Amarelo"
(integer) 3

> lrange mylist 0 -1
1) "Verde"
2) “Branco"
3) Amarelo
```

Tipo de Dado - Hash

- No Redis, um hash nada mais é do que um map que contém campos e valores do tipo string.
- Este tipo de dado é muito utilizado para representar objetos.
- Podemos definir vários conjuntos de campo-valor para uma única chave.
- Cada hash pode armazenar mais de 4 bilhões de pares de campo-valor.

Principais Comandos

- HDEL remove o(s) campo(s) e seu(s) respectivo(s) valor(es) do hash informado
- HGET chave campo — retorna o valor do campo associado ao hash informado
- HMGET retorna os valores de todos os campos informados que são associados a um hash
- HSET armazena um hash com o campo e seu respectivo valor. Caso o hash e o campo já existam, o valor é sobrescrito
- HMSET define múltiplos campos e valores em um hash
- HEXISTS determina se um hash e seu campo existem
- HLEN retorna a quantidade de campos que um hash possui
- HINCRBY incrementa o valor de um campo numérico que um hash possui

Exemplos

- Represente um perfil de usuário básico como um hash:

```
> HSET user:123 username martina firstName Martina lastName Elisa country GB
(integer) 4
> HGET user:123 username
"martina"
> HGETALL user:123
1) "username"
2) "martina"
3) "firstName"
4) "Martina"
5) "lastName"
6) "Elisa"
7) "country"
8) "GB"
```

Exemplos

- Armazene contadores para o número de vezes que o dispositivo 777 fez ping no servidor, emitiu uma solicitação ou enviou um erro:

```
> HINCRBY device:777:stats pings 1
(integer) 1
> HINCRBY device:777:stats pings 1
(integer) 2
> HINCRBY device:777:stats pings 1
(integer) 3
> HINCRBY device:777:stats errors 1
(integer) 1
> HINCRBY device:777:stats requests 1
(integer) 1
> HGET device:777:stats pings
"3"
> HMGET device:777:stats requests errors
1) "1"
2) "1"
```

Tipo de Dado - Sets

- É uma coleção não ordenada de strings exclusivas (membros).
- Podemos usar conjuntos Redis para:
  - Rastreie itens exclusivos (por exemplo, rastreie todos os endereços IP exclusivos que acessam uma determinada postagem do blog).
- Representam relações (por exemplo, o conjunto de todos os usuários com um determinado papel).
- Execute operações de conjunto comuns, como interseção, uniões e diferenças.

Principais Comandos

- SADD - adiciona um novo membro a um conjunto.
- SREM - remove o membro especificado do conjunto.
- SISMEMBER - testa uma string para associação ao conjunto.
- SINTER - retorna os elementos em comum entre os conjuntos (ou seja, a interseção).
- SDIFF - retorna os elementos que pertencem somente ao primeiro conjunto
- SUNION - retorna todos os elementos dos conjuntos informados
- SCARD - retorna o tamanho de um conjunto.

Exemplos

- Armazenar o conjunto de IDs de livros favoritos para os usuários 101 e 105:

```
> sadd usuario:101:favoritos 347
(integer) 1

> sadd usuario:101:favoritos 561
(integer) 1

> sadd usuario:101:favoritos 742
(integer) 1

> sadd usuario:105:favoritos 561
(integer) 1

> sadd usuario:105:favoritos 803
(integer) 1
```

```
> smembers usuario:101:favoritos
1) 347
2) 561
3) 742

> smembers usuario:105:favoritos
1) 561
2) 803

> scard usuario:101:favoritos
(integer) 3

> scard usuario:105:favoritos
(integer) 2
```

Exemplos

- Verificando se o usuário 101 gosta dos livros com ids 561 e 803

```
> smembers usuario:101:favoritos 561
(integer) 1

> smembers usuario:101:favoritos 803
(integer) 0
```

- Verificando os livros favoritos em comum entre os usuários 101 e 105

```
> sinter usuario:101:favoritos usuario:105:favoritos
1) 561
```

- Verificando os livros favoritos do usuário 101 que não fazem parte dos favoritos do usuário 105

```
> sdiff usuario:101:favoritos usuario:105:favoritos
1) 347
2) 742
```

Exemplos

- Remover o livro 724 dos favoritos do usuário 101:

```
> smembers usuario:101:favoritos
1) 347
2) 561
3) 742

> srem usuario:101:favoritos 724
(integer) 1

> smembers usuario:101:favoritos
1) 347
2) 561
```

Tipo de Dado - Sorted Sets

- Sorted Set é uma coleção de strings exclusivas (membros) ordenadas por uma pontuação associada.
- Exemplo de uso:
  - Podemos usar conjuntos ordenados para manter facilmente listas ordenadas das pontuações mais altas em um jogo online.

Principais Comandos

- ZADD - adiciona um novo membro e pontuação associada a um conjunto classificado. Se o membro já existir, a pontuação é atualizada.
- ZRANGE - retorna membros de um conjunto classificado, classificado dentro de um determinado intervalo.
- ZRANK - retorna a classificação do membro fornecido, supondo que a classificação esteja em ordem crescente.
- ZREVRANK - retorna a classificação do membro fornecido, supondo que o conjunto classificado esteja em ordem decrescente

Exemplos

- Atualize uma tabela de classificação em tempo real à medida que as pontuações dos jogadores mudam:

```
> ZADD leaderboard:455 100 user:1
(integer) 1
> ZADD leaderboard:455 75 user:2
(integer) 1
> ZADD leaderboard:455 101 user:3
(integer) 1
> ZADD leaderboard:455 15 user:4
(integer) 1
> ZADD leaderboard:455 275 user:2
(integer) 0
```

- Observe que a pontuação do user:2 é atualizada na última chamada ZADD.

Exemplos

- Obtenha as pontuações dos 3 melhores jogadores:

```
> ZRANGE leaderboard:455 0 2 REV WITHSCORES
1) "user:2"
2) "275"
3) "user:3"
4) "101"
5) "user:1"
6) "100"
```

PUC Minas

Exemplos

- Qual é a classificação do usuário 2?

```
> ZREVRANK leaderboard:455 user:2
(integer) 0
```

Referências

- Carlson , Josiah. Redis in Action Manning Publishing, 1ª Ed.
- Lazoti, Rodrigo. Armazenando dados com Redis, 1ª Ed.
- Redis CLI. Disponível em: https://redis.io/docs/manual/cli/