

NOSQL – GRAPH DATABASE

MARCIO JASINSKI

FURB

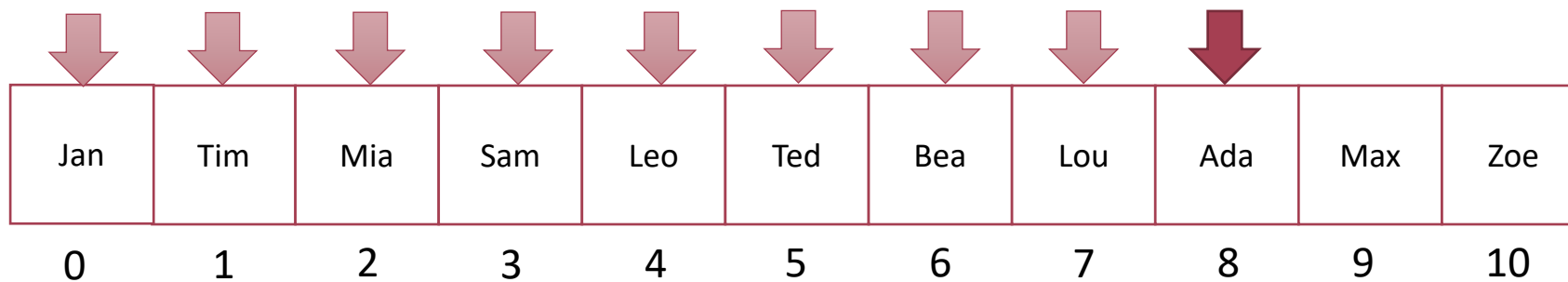
1. INTRODUÇÃO

Conceitos básicos de hash

INTRODUÇÃO

- Uma Tabela Hash é um dos conceitos básicos de um banco chave valor
- Mas como funciona uma tabela hash mesmo?
- Suponha que temos 11 “pessoas” para armazenadas.

Qual o custo de encontramos o Ada em um array normal?

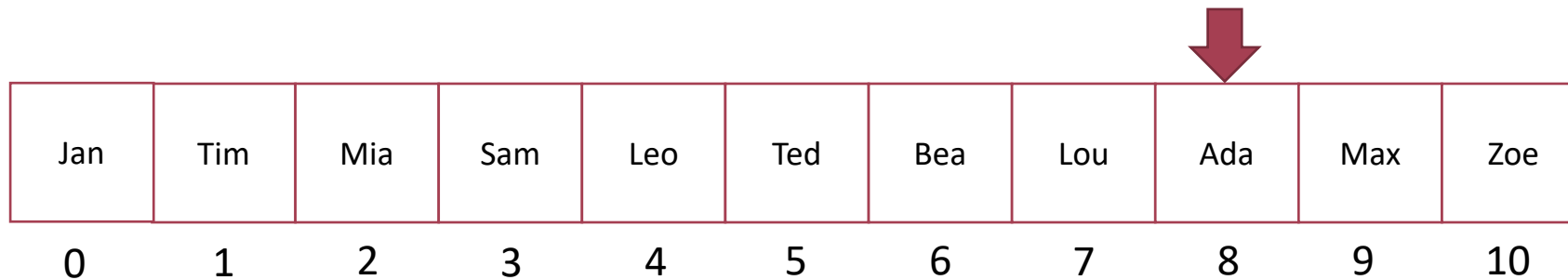


$$\boxed{\text{Ada}} = 8$$

$O(n)$

INTRODUÇÃO

- E se já soubéssemos o índice da Ada? Qual seria o custo?
- Esse é o princípio de uma tabela hash (ou tabela de espalhamento).
- Tudo parte da ideia de uma função hash...
- O conceito é fazer um cálculo pelo valor que vamos armazenar e chegar em um índice...



Jan	Tim	Mia	Sam	Leo	Ted	Bea	Lou	Ada	Max	Zoe
0	1	2	3	4	5	6	7	8	9	10

INTRODUÇÃO

- Vamos transformar nomes em números..
- Que tal a Tabela ASCII? É só um exemplo ;)
- Podemos fazer uma função simples, que soma os valores da String
- Assim **Ada = 262**
- E assim para cada String...

Sinal	<u>Dec</u>
<u>@</u>	64
<u>A</u>	65
<u>B</u>	66
<u>C</u>	67
<u>D</u>	68
<u>E</u>	69
<u>F</u>	70
<u>G</u>	71
<u>H</u>	72
<u>I</u>	73
<u>J</u>	74
<u>K</u>	75
<u>L</u>	76


INTRODUÇÃO

- Uma tabela hash utiliza faz o espalhamento das funções hash em um “array”.
- Essa operação é feita pela operação Módulo (resto da divisão) usando o tamanho da tabela

Nome	Hash	Hash Mod 11
Mia	279	4
Tim	298	1
Bea	264	0
Zoe	302	5
Jan	281	6
Ada	262	9
Leo	288	2
Sam	289	3
Lou	304	7
Max	294	8
Ted	285	10

HASHES

- Agora temos uma tabela hash onde os nomes são inseridos conforme a função
- $\text{índice} = \text{hash}(\text{nome}) \bmod 11$

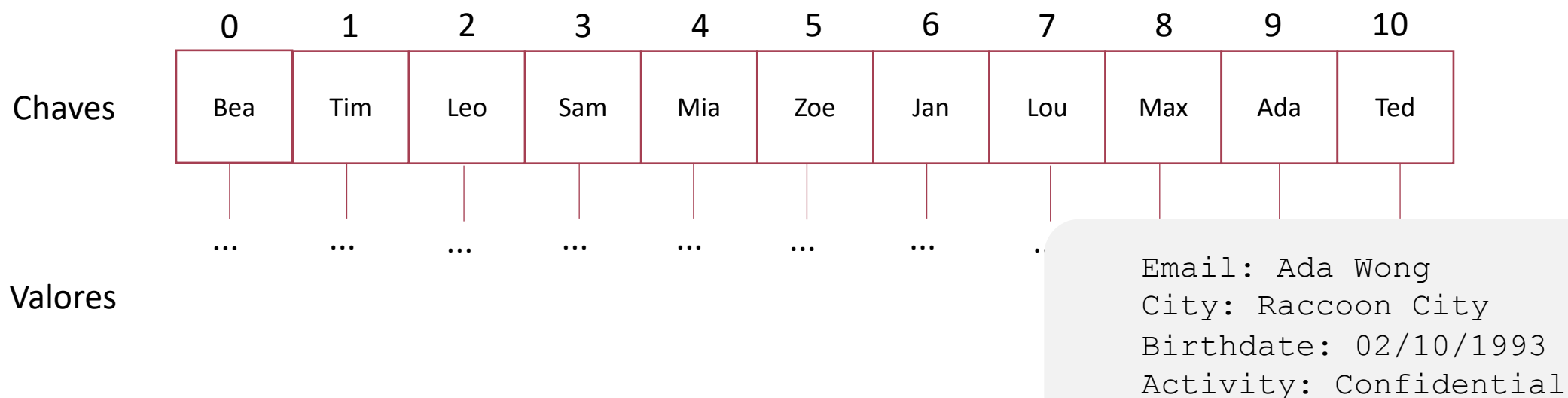


Bea	Tim	Leo	Sam	Mia	Zoe	Jan	Lou	Max	Ada	Ted
0	1	2	3	4	5	6	7	8	9	10

- Qual o custo agora para obtermos a **Ada**?
- $\text{Índice} = \text{hash}(\text{ada}) \bmod 11 \rightarrow \text{índice} = 8$
- Ou seja custo $O(1)$

KEY-VALUE DATABASES

- O princípio de um key-value database é esse e mais um modelo de distribuição das chaves valores
- A maioria dos bancos implementa um dicionário (tabela hash) distribuído entre vários servidores
- Alguns fazem isso em memória e outros de forma persistente
- Mas o resultado é sempre o mesmo, uma velocidade quase constante para obter valores a partir de uma chave



Redis

REDIS - INTRODUÇÃO

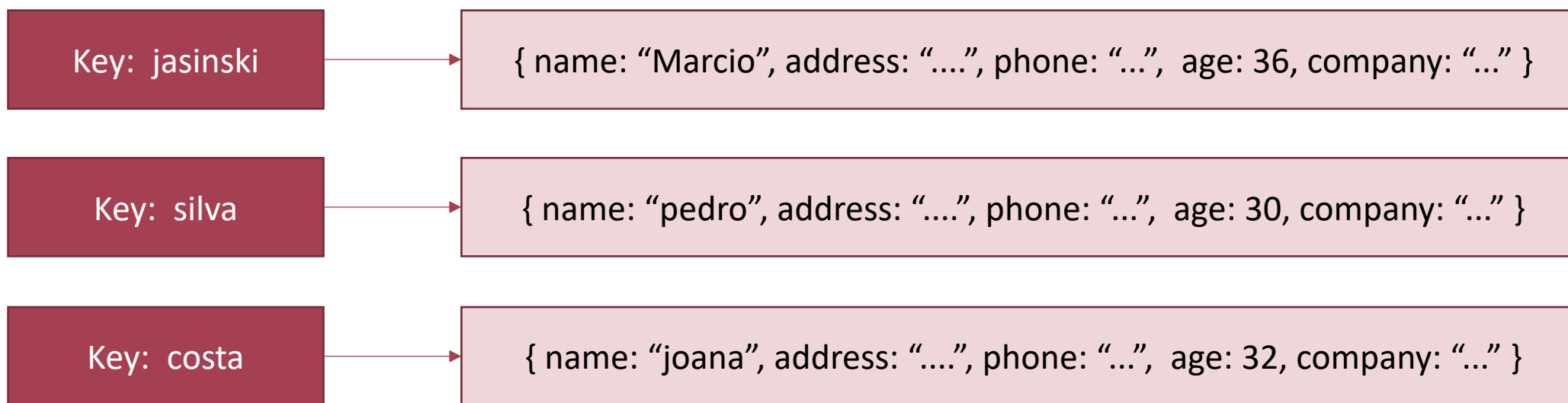
- **Redis** – (REmote DIctionary Server) é um banco de dados chave-valor em memória.
- O Redis é um tipo de banco de dados chamado de data structure server.
- É basicamente um “HashMap Database” com estruturas adicionais como listas, mapas e conjuntos (sets)
- Operações atômicas sobre as estruturas de dados
- Utilizado quando há necessidade de performance para operar com itens por uma chave

REDIS - INTRODUÇÃO

- Popularizado como parte de muitas aplicações. Principais utilizações do Redis:
- **Cache de dados** – Devido a rapidez que o banco oferece para operações de leitura e escrita, o Redis é muito usado como camada de cache de websites e aplicações que desejam compartilhar dados entre muitos servidores.
- **Publisher/Subscriber e Filas** – Desde a versão 2.0, o Redis oferece recursos de filas e modelos PubSub e por isso a solução passou a ser usada como parte de comunicação de mensageria ou tratamento assíncrono. Tem se tornado uma alternativa bem interessante a opções como RabbitMQ.
- **Contadores** – Por oferecer operações atômicas, o Redis é em geral uma excelente opção para contabilizar itens que podem ser atualizados por diversos agentes em uma rede distribuída.

REDIS – A KEY VALUE DATA STORE

- Armazenamento chave valor.



O que acontece se perder a chave?

REDIS - COMANDOS

- SET – Define um par chave => valor no redis para armazenamento
- O valor é sempre uma **string**
- Se a chave já existir, o valor será sobrescrito

Parâmetros opcionais

EX segundos – define um tempo de expiração

PX milessegundos – Define um tempo de expiração em milessegundos

NX – Só adiciona a chave/valor se não existir ainda

XX – Só adiciona a chave/valor se a mesma já existir

```
redis> SET servername "redis.furb"  
"OK"
```

```
redis> SET joeSession "20200312"  
"OK"
```

```
redis> SETNX joeSession "20200312"  
"OK"
```

REDIS - COMANDOS

- **GET** – Obtém um valor a partir de uma chave
 - Se não existir, retorna **nil**
 - Só permite retornar uma string
-
- **DEL** – Remove o par chave/valor do banco

```
redis> GET servername  
"redis.furb"
```

```
redis> GET joeSession  
"20200312"
```

```
redis> DEL servername joeSession  
"OK"
```

REDIS - COMANDOS

- **INCR** – Incrementa um número de forma atômica associado a uma chave
 - Se a chave não existir, inicializa a mesma com valor zero
 - Retorna erro se o valor existir e não for compatível com um número
-
- **INCRBY** – Incrementa conforme o parâmetro recebido de forma atômica um número associado a uma chave

```
redis> SET mykey "10"  
"OK"
```

```
redis> INCR mykey  
(integer) 11
```

```
redis> GET mykey  
"11"
```

```
redis> SET mykey "10"  
"OK"
```

```
redis> INCRBY mykey 5  
(integer) 15
```

REDIS - COMANDOS

- Por que usar o INCR se é possível fazer um contador com SET e GET?

```
SET counter 0  
total = GET counter  
total = total +  
SET counter total
```


REDIS - COMANDOS

- **INCR** – Pode ser usado como controle de quota de API. Por exemplo, uma API que não pode ser chamada mais de N vezes em 1 segundo por cliente.

```
FUNCTION LIMIT_API_CALL(ip)
ts = CURRENT_UNIX_TIME()
keyname = ip+":"+ts
current = GET(keyname)
IF current != NULL AND current > 10
    THEN ERROR "too many requests per second"
ELSE
    MULTI INCR(keyname,1)
    EXPIRE(keyname,10)
    EXEC PERFORM_API_CALL()
END
```

REDIS - COMANDOS

- **EXPIRE** – define um timeout para uma chave. Uma vez que esse timeout expira, a chave e o valor associado são automaticamente apagados.
- Chaves com timeouts associados são chamadas de chaves voláteis
- Comandos como DEL, SET e GETSET podem “limpar” o timeout definido pelo EXPIRE
- Demais comandos que atuam sobre o valor ou alteram o nome da chave não impactam no timeout
- Um novo EXPIRE sobre uma chave com timeout definido renova o tempo de expiração
- O comando TTL indica o tempo restante de uma chave.

```
redis> SET mykey "Hello"  
"OK"
```

```
redis> EXPIRE mykey 10  
(integer) 1
```

```
redis> TTL mykey  
(integer) 10
```

```
redis> SET mykey "Hello World"  
"OK"
```

```
redis> TTL mykey  
(integer) -1
```

REDIS - COMANDOS

- **EXPIRE** – define um timeout para uma chave. Uma vez que esse timeout expira, a chave e o valor associado são automaticamente apagados.
- Chaves com timeouts associados são chamadas de chaves voláteis
- Comandos como DEL, SET e GETSET podem “limpar” o timeout definido pelo EXPIRE
- Demais comandos que atuam sobre o valor ou alteram o nome da chave não impactam no timeout
- Um novo EXPIRE sobre uma chave com timeout definido renova o tempo de expiração
- O comando TTL indica o tempo restante de uma chave.

```
redis> SET mykey "Hello"  
"OK"
```

```
redis> EXPIRE mykey 10  
(integer) 1
```

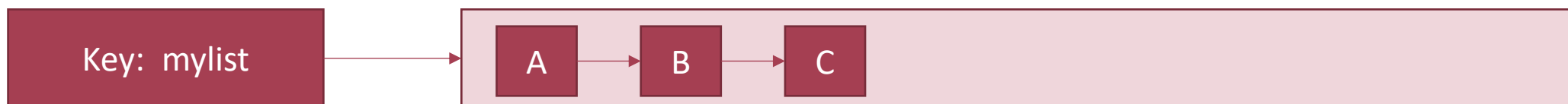
```
redis> TTL mykey  
(integer) 10
```

```
redis> SET mykey "Hello World"  
"OK"
```

```
redis> TTL mykey  
(integer) -1
```

REDIS – COMANDOS (LISTAS)

- Uma lista no Redis é uma série de valores ordenados (conforme inseridos);
- Os comandos para interagir com listas são R PUSH, L PUSH, LLEN, L RANGE, L POP, and R POP
- Listas são elementos muito flexíveis porém tendem a ter um desempenho $O(N)$ para algumas operações



REDIS – COMANDOS (LISTAS)

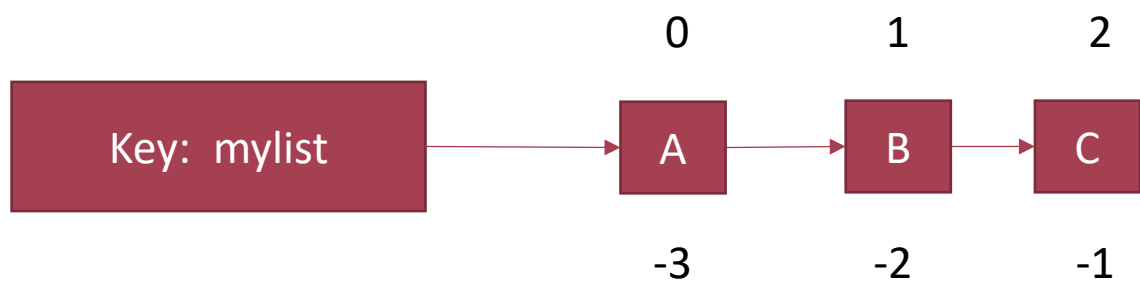
- **RPUSH** - Insere os itens especificados no final da lista associada `chave informada. Se a chave não existir, será criado o par chave/valor. Caso a chave não seja do tipo lista, retorna erro.
- **LPUSH** - Insere os itens especificados no início da lista associada `chave informada. Se a chave não existir, será criado o par chave/valor. Caso a chave não seja do tipo lista, retorna erro.

```
redis> RPUSH mylist "hello"  
(integer) 1
```

```
redis> RPUSH mylist "world"  
(integer) 2
```

REDIS – COMANDOS (LISTAS)

- **LRANGE** – retorna uma sublista conforme os parâmetros informados:
Início e Fim. Numeros negativos podem ser usados para considerar o fim da lista, por exemplo -1 ultimo, -2 penúltimo.
- **LRANGE** tem uma complexiadade $O(S+N)$ onde S é a distância do elemento inicial da busca e N é o ultimo elemento do intervalo.



```
redis> RPUSH mylist "a"  
(integer) 1
```

```
redis> RPUSH mylist "b"  
(integer) 2
```

```
redis> RPUSH mylist "c"  
(integer) 2
```

```
redis> LRANGE mylist 0 -1  
1) "a"  
2) "b"  
3) "c"
```

REDIS – COMANDOS (LISTAS)

- **LLEN** – Retorna o tamanho da lista armazenada para a chave especificada. Complexidade $O(1)$
- **RPOP** – Remove e retorna o último elemento da lista para a chave especificada
- **LPOP** - Remove e retorna o primeiro elemento da lista para a chave especificada

```
redis> LLEN mylist  
(integer) 3
```

```
redis> RPOP mylist  
"C"
```

```
redis> LPOP mylist  
"A"
```

REDIS – COMANDOS (LISTAS)

- **LSET** – Adiciona um elemento em um índice específico
- **LTRIM** – Limita a lista em um range especificado `LTRIM mylist 0 5`
- **LREM** – Remove as primeiras ocorrências do elemento conforme o parâmetro

```
redis> LSET mylist 0 "Z"  
(integer) 1
```

```
redis> LRANGE mylist 0 -1  
1) "z"  
2) "a"  
3) "b"  
4) "c"
```

```
redis> LREM mylist -2 "a"  
(integer) 1
```

“remover as ultimas duas ocorrências de
“a” da lista

REDIS – COMANDOS (SETS)

- As listas são excelentes para manipulação nas pontas e garantem a ordem de inserção. Porém, listas são lentas para identificar se um objeto pertence à estrutura e pode permitir itens duplicados.
- **Set** – é um conjunto semelhante a uma lista, mas não garante nenhuma ordem e cada elemento deve ser único.
- Comandos para manipulação de sets:
SADD, SREM, SISMEMBER, SMEMBERS and SUNION.

REDIS – COMANDOS (SETS)

- **SADD** – Adiciona o elemento especificado ao conjunto associado à chave informada.
- **SMEMBERS** - Retorna os elementos da lista especificada.

```
redis> SADD myset "a"
```

```
(integer) 1
```

```
redis> SADD myset "b"
```

```
(integer) 1
```

```
redis> SADD myset "c"
```

```
(integer) 0
```

```
redis> SADD myset "a"
```

```
(integer) 0
```

```
redis> SMEMBERS myset
```

```
1) "c"
```

```
2) "b"
```

```
3) "a"
```

REDIS – COMANDOS (SETS)

- **SREM** – Remove o item do conjunto.
- **SISMEMBER** – Indica se o item especificado existe no conjunto ou não

```
redis> SREM myset "a"  
(integer) 1
```

```
redis> SREM myset "b"  
(integer) 1
```

```
redis> SISMEMBER myset "c"  
(integer) 1
```

```
redis> SMEMBERS myset  
1) "c"
```

REDIS – COMANDOS (SETS)

- **SPOP** – Remove um ou mais elementos de forma aleatória e remove ele do conjunto
- **SRANDMEMBER** – retorna um elemento aleatório sem removê-lo do conjunto

```
redis> SADD myset "one"  
(integer) 1
```

```
redis> SADD myset "two"  
(integer) 1
```

```
redis> SADD myset "three"  
(integer) 1
```

```
redis> SPOP myset  
"one"
```

REDIS – COMANDOS (SORTED SETS)

- **Sorted Set** – É um tipo especial de conjunto que associa um score a cada elemento. Assim, fica fácil de ordenar os elementos. Os elementos continuam sendo únicos, mas o score pode se repetir.

```
redis> ZADD heroesRank 1960 "Batman"  
(integer) 1
```

```
redis> ZADD heroesRank 1954 "Spiderman"  
(integer) 1
```

```
redis> ZADD heroesRank 1983 "Ironman"  
(integer) 1
```

```
redis> ZRANGE heroesRank 0 3 WITHSCORES  
1) "Spiderman"  
2) "Batman"  
3) "Ironman"
```

REDIS – COMANDOS (SORTED SETS)

- **ZREM** – Remove um item do conjunto
- **ZRANK** – Retorna o índice do conjunto para um dado item

```
redis> ZRANK heroes "Batman"  
(integer) 1
```

```
redis> ZRANK heroes "Spiderman"  
(integer) 0
```

```
redis> ZRANK heroesRank 1983 "Ironman"  
(integer) 2
```

REDIS – COMANDOS (SORTED SETS)

- **ZREM** – Remove um item do conjunto
- **ZRANK** – Retorna o índice do conjunto para um dado item

```
redis> ZRANK heroes "Batman"  
(integer) 1
```

```
redis> ZRANK heroes "Spiderman"  
(integer) 0
```

```
redis> ZRANK heroesRank 1983 "Ironman"  
(integer) 2
```

REDIS – COMANDOS (SORTED SETS)

- **ZINCRBY** – Incrementa de forma atômica o elemento especificado
- **ZPOPMAX** – Remove e retorna o elemento com maior score do conjunto
- **ZPOPMIN** – Remove e retorna o elemento com menor score do conjunto
- **ZUNIONSTORE** - Permite unir dois sorted sets
- **ZSCORE** – retorna o score do item especificado

REDIS – COMANDOS (HASHES)

- **Hashes** - São estrutura chave/valor que permitem armazenada diversos campos com valores que se assemelham muito a demanda de armazenada objetos
- **HSET** – Comando para definir pares chave/valor em uma chave do Redis

```
redis> HSET user:1000 name "Marcio Jasinski"
```

```
(integer) 1
```

```
redis> HSET user:1000 email "marciogj@gmail.com"
```

```
(integer) 1
```

```
redis> HSET user:1000 username "marcio.jasinski"
```

```
(integer) 1
```

```
redis> HGETALL user:1000
```

```
1) "name"
```

```
2) "Marcio Jasinski"
```

```
3) "email"
```

```
4) marciogj@gmail.com
```

```
...
```

REDIS – COMANDOS (HASHES)

- **HGET** – Retorna uma propriedade específica informada no comando
- **HINCRBY** – Incrementa de forma atômica um elemento de um tipo hash
- **HLEN** – Informa a quantidade de hashes nesse elemento
- **HSTRLEN** – Retorna o tamanho de uma string de uma chave pertencente ao element informado.
- **HKEYS** – Retorna as chavaes associadas ao element informado.
- **HEXISTS** – Verifica se o hash existe para a chave informada.
- **HDEL** – remove um hash

REDIS – COMANDOS (HASHES)

- **HGET** – Retorna uma propriedade específica informada no comando
- **HINCRBY** – Incrementa de forma atômica um elemento de um tipo hash
- **HLEN** – Informa a quantidade de hashes nesse elemento
- **HSTRLEN** – Retorna o tamanho de uma string de uma chave pertencente ao element informado.
- **HKEYS** – Retorna as chavaes associadas ao element informado.
- **HEXISTS** – Verifica se o hash existe para a chave informada.
- **HDEL** – remove um hash

REDIS – COMANDOS (MULTI)

- **MULTI** – Marca o início de um processo transacional no Redis. Todos os comandos subsequentes serão enfileirados para execução única com o comando EXEC.
- **EXEC** – Executa os comandos enfileirados e retorna o status da conexão para normal.

Atividade

EXERCÍCIOS

- Instalação do Redis
- Utilizar a linguagem que desejar e implementar um jogo de bingo com o Redis (ou Redinsgo :D)
- Ver material de apoio.



Marcio.Jasinski@senior.com.br