Redis com Java e Python

Fernando Anselmo
http://fernandoanselmo.orgfree.com/wordpress/

Versão 1.1 em 24 de outubro de 2021

Resumo

Redis é um banco do tipo NoSQL de Chave-Valor que pode ser utilizado tanto como banco de dados, bem como mecanismos de cache e agente de mensagens. O Redis fornece uma estruturas de dados como strings, hashes, listas e conjuntos. Sendo que os conjuntos podem ser classificados com consultas de intervalo, bitmaps, hiperloglogs, índices geoespaciais e fluxos. O Redis tem replicação integrada, script Lua, despejo de LRU, transações e diferentes níveis de persistência em disco. Ainda por cima oferece uma alta disponibilidade por meio do Redis Sentinel e particionamento automático com Redis Cluster. Neste tutorial veremos o que vem a ser o banco NoSQL Redis [1] e como proceder sua utilização utilizando como pano de fundo a linguagem de programação Java [2] e Python [3].

1 Parte inicial

Redis é um dos mais rápidos bancos NoSQL conhecidos, possui tempos de resposta abaixo um milissegundo, permite milhões de solicitações por segundo para aplicativos em tempo real de setores como jogos, tecnologia de publicidade, serviços financeiros, saúde e IoT. Atualmente é considerado um dos mais populares para mecanismos de cacheamento, ganhou o título do banco de dados "Mais Amado"pelo Stack Overflow por cinco anos consecutivos. Por causa de seu rápido desempenho, é a escolha popular para armazenamento em cache, gerenciamento de sessão, placares e estatística de jogos, análises em tempo real, informação geoespacial, sistemas de bate-papo e mensagens e streaming de mídia.

Figura 1: *Logo do Redis*

Todos os dados do Redis residem na memória, o que permite acesso a dados de baixa latência e alto rendimento. Ao contrário dos bancos de dados tradicionais, os armazenamentos de dados na memória não exigem uma leitura física ao disco, reduzindo a latência do mecanismo para microssegundos. Por causa disso, os armazenamentos de dados na memória podem suportar uma ordem de magnitude a mais de operações e tempos de resposta mais rápidos. O resultado é um desempenho extremamente rápido com operações médias de leitura e gravação levando menos de um milissegundo e suporte para milhões de operações por segundo.

Ao contrário de outros armazenamentos no padrão chave-valor, o Redis possui uma grande variedade de estruturas de dados para atender às necessidades de seu aplicativo. Os tipos de dados Redis incluem:

- **Strings** - texto ou dados binários de até 512 MB de tamanho.
- **Listas** - uma coleção de Strings na ordem em que foram adicionadas.
- **Conjuntos** - uma coleção não ordenada de strings com a capacidade de cruzar, unir e diferenciar outros tipos de Conjunto.
- **Conjuntos classificados** - conjuntos ordenados por um valor.
- **Hashes** - destinado a armazenar uma lista de chaves e valores.
- **Bitmaps** - um tipo de dados que oferece operações a nível de bits.
- **HyperLogLogs** - uma estrutura de dados probabilística para estimar os itens únicos em um conjunto de dados.
- **Streams** - uma estrutura de dados de registro Fila de mensagens. Geoespacial - uma entrada baseada em longitude / latitude Mapas, "nas proximidades".

1.1 Criar o contêiner Docker

A forma mais simples para obtermos o Redis é através de um contêiner no Docker, deste modo podemos ter várias versões do banco instalada e controlar mais facilmente qual banco está ativo ou não. E ainda colhemos o benefício adicional de não termos absolutamente nada deixando sujeira em nosso sistema operacional ou áreas de memória.

Baixar a imagem oficial:

```
$ docker pull redis
```

Antes de criarmos o contêiner precisamos criar uma pasta em nosso disco para hospedar o banco, por exemplo minha pasta chama-se: "/home/fernando/redis-data". Deste modo a criação do contêiner será:

```
$ docker run -p 6379:6379 -v /home/fernando/redis-data:/data --name meu-redis
-d redis redis-server --appendonly yes
```

Acessar o **redis-cli** no contêiner:

```
$ docker exec -it meu-redis redis-cli
```

Verificamos informações da base de dados:

```
> info
```

Ou retornamos ao terminal:

```
> exit
```

Deste modo podemos executar quaisquer comandos que veremos na seção **Redis-Cli**. Paramos o contêiner:

```
$ docker stop meu-redis
```

E reiniciamos novamente:
`$ docker start meu-redis`

1.2 Gerenciar o Redis

Para realizar o gerenciamento do Redis optamos por uma interface gráfica conhecida como "*Another Redis Desktop Manager*"[4] que é um projeto no GitHub criado por Qishibo, e para o Linux a forma mais prática para instalar é baixar uma AppImage que é simplesmente fazer download e torná-la um executável. Esta pode ser encontrada neste endereço: `https://appimage.github.io/Another_Redis_Desktop_Manager/`

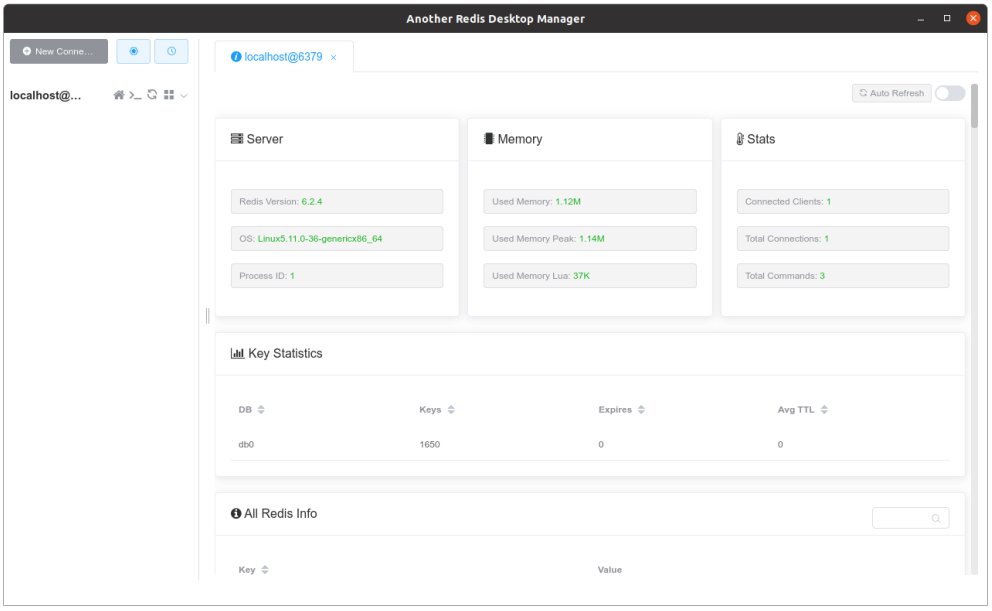

Figura 2: *Tela Principal do Another Redis Desktop Manager*

Se não pode usar AppImage em seu sistema ou se preferir pode realizar um clone do repositório, mas lembrando que para isso é necessário ter o npm instalado:
`$ git clone https://github.com/qishibo/AnotherRedisDesktopManager.git`
`$ cd AnotherRedisDesktopManager`

Instalar as dependências:
`$ npm install`

DICA: Se falhou para instalar o Electron durante a instalação utilize o seguinte comando:
`$ ELECTRON_MIRROR="https://npm.taobao.org/mirrors/electron/"npm install`

E inicie a aplicação que estará na "localhost:9988":
`$ npm start`

Para iniciar o cliente basta usar o electron com o seguinte comando:
`$ npm run electron`

Este é um aplicativo extremamente simples, basta realizarmos uma conexão com o banco redis indicando seu endereço e porta, teremos todos os dados do servidor, memória, conexões, estatísticas e completas informações sobre o banco. Ao se posicionar em uma chave gravada (um registro)

podemos ver sua estrutura, modificar ou eliminar seus valores além de realizar pesquisas.

2 Redis-Cli

Diferente dos outros bancos o Redis preza por ser extremamente simples, podemos selecionar qualquer base com o seguinte comando:

```
> select N
```

Onde este N é simplesmente o número da base, a primeira inicia com 0, sendo assim:

```
> select 0
```

Este comando verifica a quantidade de elementos:

```
> dbsize
```

Outro simples comando para saber que está tudo OK é:

```
> ping
```

E a resposta deve ser **PONG** o que significa "estamos sem problemas". Para inserirmos um simples elemento:

```
> set valor 100
```

E para resgatá-lo:

```
> get valor
```

Qualquer tipo para o Redis pode ser tratado como uma String, assim para obtermos o tamanho de uma determinada chave:

```
> strlen valor
```

Mas observamos também que por ser um número podemos realizar operações como incrementar o valor:

```
> incr valor
```

Ou decrementar o valor:

```
> decr valor
```

O mais curioso é que podemos ainda concatenar (anexar) mais informações:

```
> decr 200
```

E agora nossa chave tem o valor "100200". E podemos adicionar mais valores:

```
> incrby valor 50
```

Ou diminuir valores:

```
> decrby valor 50
```

2.1 Chave-Valor

O maior uso do Redis e o armazenamento de informações como uma tabela no formato chaves-valores:

```
> hmset identificacao nome Fernando cargo Analista materia Estatística
```

Adicionar mais uma propriedade:

```
> hset identificacao cidade Brasília
```

Recuperar uma determinada propriedade:
```
> hmget identificacao cargo
```

Obter todos os valores:
```
> hvals identificacao
```

Obter todos os nomes das chaves:
```
> hkeys identificacao
```

Ou mais completo como:
```
> hgetall identificacao
```

Obter o tamanho:
```
> hlen identificacao
```

Eliminar uma propriedade:
```
> hdel identificacao materia
```

2.2 Listas de valores

Para criar uma chave como valores de lista e adicionar elementos a ela (a direita ou no final):
```
> rpush tipos persa ragdoll bengali
```

Ou então (a esquerda ou no início):
```
> lpush tipos siamês
```

Recuperar todos os tipos (se ainda não percebeu de gatos):
```
> lrange tipos 0 -1
```

Ou um determinado tipo (por exemplo o terceiro elemento - a lista inicia do 0):
```
> lrange tipos 2 2
```

Verificar o tamanho da lista:
```
> llen tipos
```

Eliminar determinado elemento:
```
> lpop tipos 2
```

2.3 Conjuntos de valores

Para criarmos uma chave como valores de conjuntos e adicionar elementos a ela:
```
> sadd paises Brasil EUA Portugal Espanha
```

Recuperar a quantidade de países:
```
> scard paises
```

Recuperamos todos os países:
```
> smembers paises
```

Ou obter um país aleatoriamente:
```
> srandmember paises 1
```

Eliminar determinado elemento:
```
> srem paises 2
```

2.4 Movimentos e Exclusão

Vericar se uma determinada chave existe:
```
> exists paises
```

Mover uma determinada chave para outro banco:
```
> move tipos 1
```

Modificar o nome de uma determinada chave:
```
> rename paises destinos
```

Para excluirmos qualquer chave temos mais um comando extremamente simples:
```
> del destinos
```

3 Linguagem Java

Java é considerada a linguagem de programação orientada a objetos mais utilizada no Mundo, base para a construção de ferramentas como Hadoop, Pentaho, Weka e muitas outras utilizadas comercialmente. Foi desenvolvida na década de 90 por uma equipe de programadores chefiada por *James Gosling* para o projeto Green, na Sun Microsystems - tornou-se nessa época como a linguagem que os programadores mais baixaram e o sucesso foi instantâneo. Em 2008 o Java foi adquirido pela Oracle Corporation.

3.1 Driver JDBC de Conexão

Para proceder a conexão com Java, é necessário baixar um driver JDBC (Java Database Connection). Existem vários drivers construídos. Para utilizar o driver é necessário criar um projeto (vamos usar o **Spring Tool Suite 4**, utilize se quiser qualquer outro editor de sua preferência).

No STS4 acessar a seguinte opção no menu: File ▷ New ▷ Java Project. Informar o nome do projeto (Decus), não esquecer de modificar a opção "Use an environment JRE"para a versão correta da Java Runtime desejada e pressionar o botão Finish.

Agora devemos convertê-lo para um projeto Apache Maven. Clicar com o botão direito do mouse no projeto e acessar a opção: Configure ▷ Convert to Maven Project. Na janela apenas pressione o botão *Finish*. Se tudo está correto observamos que o projeto ganhou uma letra **M** o que indica agora é um projeto padrão Maven. Então foi criado um arquivo chamado **pom.xml**.

Acessar este arquivo e antes da tag BUILD, inserir a tag DEPENDENCIES:

```
1 <dependencies>
2   <dependency>
3     <groupId>redis.clients</groupId>
4     <artifactId>jedis</artifactId>
5     <version>2.8.1</version>
6   </dependency>
```

```
</dependencies>
```

Agora a situação do projeto é esta:

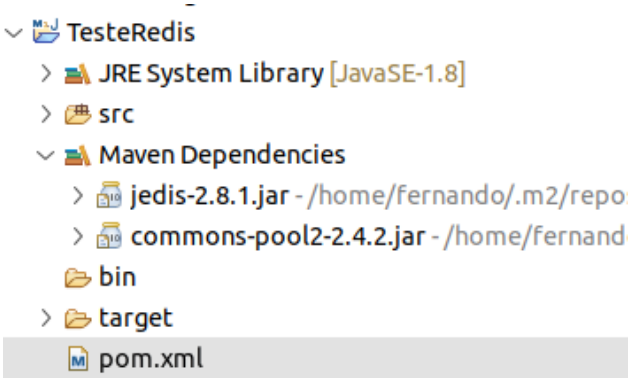

Figura 3: *Dependências do Maven*

Observamos que na pasta **Maven Dependencias** foi baixado a versão 2.8.1 do driver Jedis - Isso não é erro de digitação realmente o driver se escreve com "J".

3.2 Exemplo Prático de uma Escola

Estamos prontos para testarmos a conexão entre Redis e Java. Criamos um pequeno exemplo que nos auxiliará como teste, uma classe chamada **Escola** no pacote **decus.com** e inserimos nesta a seguinte codificação:

```
package decus.com;

import java.text.SimpleDateFormat;
import java.util.Date;
import redis.clients.jedis.Jedis;

public class Escola {

 private String [] materias = new String[] {
  "Matemática Computacional", "Programação", "Estatística",
  "SQL", "NoSQL", "Java", "Python", "Análise de Dados",
  "Arquitetura de DW", "Big Data"
 };
 public static void main(String [] args) {
  new Escola().executar();
 }
 private void executar() {
  adicionarNotas();
 }
}
```

Nossa escola promove cursos mensais e temos um conjunto contendo todas as matérias que são aplicadas aos alunos. Ao término do curso é registrado o nome do aluno, a matéria e sua nota. Escolhemos o Redis para conter esses dados exatamente devido a sua característica de chave-valor.

No método principal que chama o executar, em Java não devemos programar no método "main". No executar colocamos uma chamada ao método criarAlunos() que terá a seguinte codificação:

```
private void adicionarNotas() {
 Jedis jedis = new Jedis();
 String data = new SimpleDateFormat("dd/MM/yyyy").format(new Date());
 String chave = "";
 String [] nomes = new String[] {
  "Alice", "Miguel", "Sophia", "Arthur", "Helena", "Bernardo", "Valentina",
  "Heitor", "Laura", "Davi", "Isabella", "Lorenzo", "Manuela", "Théo", "Júlia",
  "Pedro", "Heloísa", "Gabriel", "Luiza", "Enzo", "Maria", "Luiza", "Matheus",
  "Lorena", "Lucas", "Lívia", "Benjamin", "Giovanna", "Nicolas", "Maria",
  "Eduarda", "Guilherme", "Beatriz", "Rafael", "Clara", "Joaquim", "Cecília"
 };
 String nome = "";
 for (int i = 0; i < 50; i++) {
  nome = nomes[(int)(Math.random()*nomes.length)] + " " +
    nomes[(int)(Math.random()*nomes.length)];
  for (int j = 0; j < materias.length; j++) {
   chave = "MAT" + i + "-" + j + "-" + data;
   jedis.hset(chave, "nome", nome);
   jedis.hset(chave, "materia", materias[j]);
   jedis.hset(chave, "nota", ""+(int)(Math.random()*11));
   jedis.close();
  }
 }
 System.out.println("Notas Adicionadas");
}
```

Obviamente como isto é um simples exemplo vamos adicionar as notas de uma forma totalmente aleatória.

Neste método temos a definição do objeto do Jedis que é responsável pela comunicação com o banco de dados que deve estar disponível na porta padrão. Criamos duas listas de nomes e matérias apenas para compor as notas que iremos inserir e ter dados mais elaborados. Teremos 50 alunos que ganharão nomes e sobrenomes completamente aleatórios com base na primeira lista, para cada um deles percorremos a lista de matéria e definimos uma chave que será composta por: "MAT"+ valor de i (que é o número do aluno) + valor de j (que é o número da materia) e a data que estamos inserindo a informação. Assim temos a garantia que essa chave será única.

Para inserir a informação no objeto Jedis temos o método "hset"que recebe 3 parâmetros a chave, o nome do campo e o valor deste, e assim inserimos os campos nome, matéria e nota que será obtida também de forma aleatória (entre o valor de 0 a 10). Uma vez executado temos como resposta na console a mensagem "Notas Adicionadas"se verificamos no aplicativo "Another Redis Desktop Manager"teremos os seguintes dados:

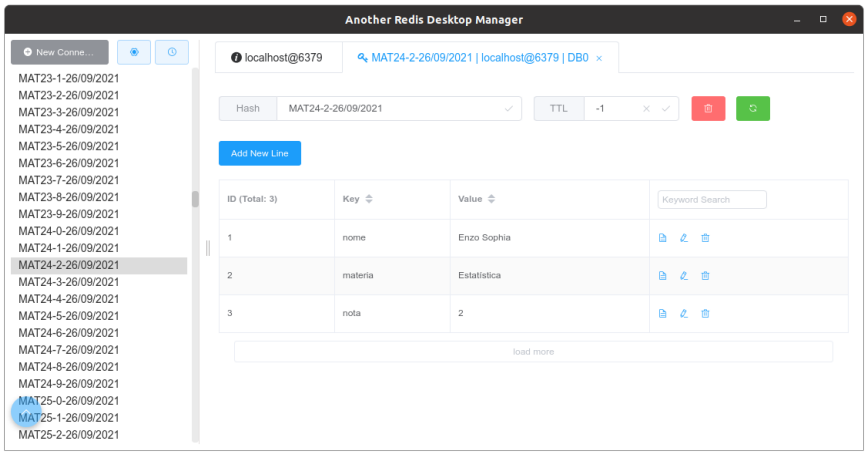

Figura 4: *Another Redis Desktop Manager*

Obviamente que os dados podem dar diferença pois o nome e as notas foram selecionados randomicamente.

3.3 Localizações

No método executar comentamos a chamada ao método "adicionarNotas()"e criamos outra para o método "trazerAluno(1)"que terá a seguinte codificação:

```
1  private void trazerAluno(int mat) {
2   Jedis jedis = new Jedis();
3   Set<String> nomes = jedis.keys("MAT" + mat + "-*");
4   java.util.Iterator<String> it = nomes.iterator();
5   String s;
6   while (it.hasNext()) {
7    s = it.next();
8    System.out.print(jedis.hget(s, "nome") + " - ");
9    System.out.print(jedis.hget(s, "materia") + " - ");
10   System.out.println(jedis.hget(s, "nota"));
11  }
12  jedis.close();
13 }
```

As coisas no Redis são bem fáceis e rápidas, recebemos a matrícula e obtemos um conjunto (objeto Set) com todas as chaves que começam pela expressão "MAT1-"(como o valor que passamos foi 1) podemos utilizar esse método para realizar qualquer tipo de pesquisa, por exemplo "*-26/09/2021"trará todas as chaves do dia "26/09/2021".

Percorremos esse conjunto e obtemos os dados que desejamos através do método "hget"no qual passamos a chave e o campo que desejamos. Ao executarmos teremos na console o nome do aluno, o nome da matéria e a nota obtida por esse.

Mas e se por exemplo desejamos saber a média geral das matérias dessa turma em cada matéria? Comentamos a chamada ao "trazerAluno(1)"no método executar por "mediaGeral()"e adicionamos a seguinte codificação:

```
1 private void mediaGeral() {
```

```
 Jedis jedis = new Jedis();
 String materia = "";
 for (int i = 0; i < materias.length; i++) {
  int totNota = 0;
  int qtdNota = 0;
  Set<String> nomes = jedis.keys("*-"+i+"-*");
  java.util.Iterator<String> it = nomes.iterator();
  String s;
  while (it.hasNext()) {
   s = it.next();
   materia = jedis.hget(s, "materia");
   totNota += Integer.parseInt(jedis.hget(s, "nota"));
   qtdNota++;
  }
  System.out.println(materia + " - " + (totNota/qtdNota));
 }
 jedis.close();
}
```

E temos na saída da console cada uma das matérias e sua respectiva média.

3.4 Eliminar Registro

Assim como a consulta do registro é obtida através da chave o mesmo se aplica para a eliminação do mesmo, vamos comentar a chamada ao método "mediaGeral()"e chamar o método "eliminarAluno(1)"e adicionarmos a seguinte codificação:

```
private void eliminarAluno(int mat) {
 Jedis jedis = new Jedis();
 Set<String> nomes = jedis.keys("MAT" + mat + "-*");
 java.util.Iterator<String> it = nomes.iterator();
 String s;
 while (it.hasNext()) {
  s = it.next();
  jedis.del(s);
 }
 jedis.close();
 System.out.println("Notas Eliminadas");
}
```

Localizamos as chaves do aluno e executamos o método "del(chave)"para cada registro. Ou seja, extremamente simples e prático.

3.5 Estrutura do objeto Jedis

Aqui utilizamos o modelo estrutural de um Hash com mapeamento de um campo String como no de campo e seu respectivo valor e que pode ser aplicado para a grande maioria dos casos, porém existem outras estruturas no Redis que podem ser utilizadas com este conector.

String

É o modelo mais simples uma chave e um valor, pode ser utilizado por exemplo para uma associação

de palavras como localidade e endereço:

```
jedis.set("Joaquim Lucas", "Qd 28 Casa 47 Rua Amélia Cidade Vale Alegre");
jedis.set("Nicolas Cecília", "Qd 7 Casa 7 Rua Sete da Cidade Amarela");
System.out.println(jedis.get("Joaquim Lucas"));
```

Ou seja, utilizamos o método "set(chave, valor)"para inserirmos a informação e o método "get(chave)"para obtê-la. Esse modelo também pode ser utilizado como uma camada de cache extremamente rápida e simples de se usar para solicitações HTTP recebidas de um aplicativo Web ou outros requisitos de cache.

Listas

São listas de strings, classificadas por ordem de inserção e podem ser uma ferramenta ideal para implementar, por exemplo, filas de mensagens:

```
jedis.lpush("tarefas", "Primeira tarefa do dia");
jedis.lpush("tarefas", "Segunda tarefa do dia");
jedis.lpush("tarefas", "Terceira tarefa do dia");
```

A cada comando "lpush(chave, valor)"uma nova tarefa é inserida. Para retirar usamos o comando:

```
System.out.println(jedis.rpop("tarefas"));
```

Devemos lembrar que precisamos serializar qualquer objeto e persisti-lo como uma string, portanto, as mensagens na fila podem transportar dados mais complexos quando necessário.

Conjuntos

Trata de uma coleção não ordenada de Strings e que são bem úteis quando desejamos excluir membros repetidos, para adicionar elementos usamos:

```
jedis.sadd("frutas", "abacate");
jedis.sadd("frutas", "manga");
jedis.sadd("frutas", "abacate");

Set<String> conjunto = jedis.smembers("frutas");
System.out.println(jedis.sismember("frutas", "abacate"));
```

As frutas do Java Set terão um tamanho de 2, a segunda adição de "abacate"foi ignorada. O método sismember permite que verifiquemos a existência de um membro específico rapidamente e mesmo com a resposta "true"se olharmos no nosso banco (usemos o aplicativo "Another Redis Desktop Manager") veremos que só existe uma única ocorrência para "abacate".

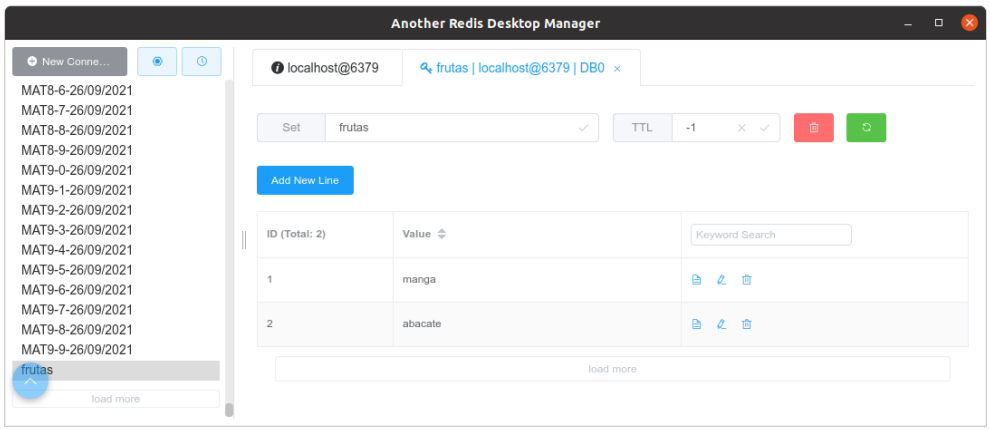

Figura 5: *Visão do Conjunto Frutas*

Conjuntos Ordenados São conjuntos classificados em que cada membro possui uma determinada classificação associada, que é usada para localizá-los. Podemos inserir alguns alunos e seus desempenhos da seguinte forma:

```
Map<String, Double> desempenho = new HashMap<>();
desempenho.put("Aluno 1", 3000.0);
desempenho.put("Aluno 2", 1500.0);
desempenho.put("Aluno 3", 8200.0);
desempenho.put("Aluno 4", 2500.0);
desempenho.put("Aluno 5", 7500.0);
desempenho.put("Aluno 6", 6200.0);
desempenho.entrySet().forEach(playerScore -> {
  jedis.zadd("Desempenho", playerScore.getValue(), playerScore.getKey());
});
```

E obtermos uma relação dos 3 melhores com a seguinte codificação:

```
System.out.println("1o.: " + jedis.zrevrange("Desempenho", 0, 1).iterator().next());
System.out.println("2o.: " + jedis.zrevrange("Desempenho", 1, 2).iterator().next());
System.out.println("3o.: " + jedis.zrevrange("Desempenho", 2, 3).iterator().next());
```

Podemos utilizar de todas essas estruturas para criarmos uma aplicação extremamente veloz ou mesmo usar o Redis como um complementar para determinadas ações que precisamos aumentar os ganhos de velocidade.

4 Python

Python é uma linguagem de programação de alto nível, interpretada a partir de um script, Orientada a Objetos e de tipagem dinâmica. Foi lançada por Guido van Rossum em 1991. Não pretendo nesta apostila COMPARAR essa linguagem com Java (espero que nunca o faça), fica claro que os comandos são bem mais fáceis porém essas linguagens possuem diferentes propósitos.

Todos os comandos descritos abaixo foi utilizado no JupyterLab [6], então basta abrir um Notebook e digitá-los em cada célula conforme se apresentam.

4.1 Pacote Necessário

Baixar o pacote necessário:
`!pip install redis`

Importar os pacotes necessários:
`import redis`

Criamos um objeto da classe Redis para realizarmos nossas manipulações:
`r = redis.Redis()`
ou então:
`r = redis.Redis(host= 'localhost',port= '6379')`

Devemos nos atentar que sempre podemos obter todas as chaves criadas com o comando:
`r.keys()`

4.2 Padrão simples Chave-Valor

E usamos o método **mset** para darmos entrada nas informações:

```
r.mset({
  "AC":"Rio Branco", "AL":"Maceió", "AP":"Macapá",
  "AM":"Manaus", "BA":"Salvador", "CE":"Fortaleza", "DF":"Brasília",
  "ES":"Vitória", "GO":"Goiânia", "MA":"São Luís", "MT":"Cuiabá",
  "MS":"Campo Grande", "MG":"Belo Horizonte", "PA":"Belém", "PB":"João Pessoa",
  "PR":"Curitiba", "PE":"Recife", "PI":"Teresina", "RJ":"Rio de Janeiro",
  "RN":"Natal", "RS":"Porto Alegre", "RO":"Porto Velho", "RR":"Boa Vista",
  "SC":"Florianópolis", "SP":"São Paulo", "SE":"Aracaju", "TO":"Palmas"})
```

Para consultar qualquer capital brasileira com o método **get(chave)**:
`r.get("PB").decode("UTF8")`

Usamos o padrão de decodificação "UTF-8"de forma que os acentos apareçam corretamente. Poderíamos também criar separadamente cada uma das 27 capitais com o método:
`r.set(chave, valor)`

Mas obviamente daria muito mais trabalho. Para verificar a existência de qualquer chave, utilizamos o método: `r.exists("PA")`

Que nos devolve o valor 1 caso a chave exista ou 0 caso contrário.

4.3 Tempo de vida da chave

Para criarmos uma chave com um tempo determinado de expiração (em microssegundos):
`r.psetex(chave, tempo_ms, valor)`

Obter o tempo da chave:
`r.pttl(name)`

Ou mesmo removê-lo completamente:
`r.persist(chave)`

4.4 Conjuntos

Conjuntos são criados da seguinte forma:

```
frutas = {"abacate", "manga", "abacate"}
r.sadd("frutas", *frutas)
```

E obtidos:
`r.smembers("frutas")`

4.5 Conjuntos Ordenados

Basicamente trabalha de mesma forma mantendo uma lista de valores ordenados no seguinte formato:

```
r.zadd('desempenho', {'Aluno 1': 56.3})
r.zadd('desempenho', {'Aluno 2': 46.2})
r.zadd('desempenho', {'Aluno 3': 67.3})
r.zadd('desempenho', {'Aluno 4': 51.7})
```

E para trazemos uma listagem inversa de valores:
`print(r.zrange('desempenho', 0, -1))`

4.6 Hash

Já vimos este formato com Java, e também podemos aplicá-lo ao Python, ou seja, se desejarmos criar o mesmo programa que vimos anteriormente (por isso que dissemos que comparações são totalmente desnecessárias):

```
r.hset("camisa", "modelo", "social")
r.hset("camisa", "tamanho", "P")
r.hset("camisa", "valor", 23.50)
```

E obtermos com:
`r.hget("camisa", "modelo")`

Outra forma é como um bloco de informação, também conhecido como *Pipeline*, observemos o seguinte código:

```
import random

random.seed(444)
camisas = {f"camisas:{random.getrandbits(32)}": i for i in (
   {
      "cor": "preto",
      "preço": 49.99,
      "qtd": 1000
   },{
      "cor": "marom",
      "preço": 59.99,
      "qtd": 500
   },{
```

```
        "cor": "verde",
        "preço": 99.99,
        "qtd": 200
    })
}
with r.pipeline() as pipe:
    for h_id, camisa in camisas.items():
        for item, valor in camisa.items():
            pipe.hset(h_id, item, valor)
        pipe.execute()
    r.bgsave()
```

E podemos localizar uma chave qualquer através do comando:
`print(r.hgetall(chave))`

Que nos devolve um objeto do tipo dicionário.

5 Conclusão

O Redis nos permite escrever códigos tradicionalmente complexos com menos linhas e de modo mais simplificado. Utilizamos menos código para armazenar e obter os dados nos aplicativos. A diferença é que podemos usar estruturas de comandos mais simplificadas em oposição às linguagens tradicionais de consulta (como o SQL). Por exemplo, podemos usar a estrutura de dados hash do Redis para mover de uma base para outra com apenas uma linha de código. Uma tarefa semelhante exigiria muitas linhas para converter de um formato para outro.

O Redis vem com estruturas de dados nativas e muitas opções para manipulação e interação. Existem ainda muitos clientes de código aberto disponíveis. Redis é um banco de dados remoto "*in-memory*"que oferece um alto desempenho, replicação e um modelo de dados exclusivo para produzir uma plataforma de armazenamento simplificado para resolver problemas comuns.

As principais linguagens de programação possuem suporte e aqui vimos apenas Java e Python, porém existem muitas outras como PHP, C, C++, C#, JavaScript, Node.js, Ruby, R e Go. Esta apostila faz parte da série dos quatro tipos para Bancos de Dados no padrão NoSQL que estou tentando desmistificar e torná-los mais acessíveis tanto para as comunidades de Java e Python voltada especificamente para desenvolvedores ou cientistas de dados.

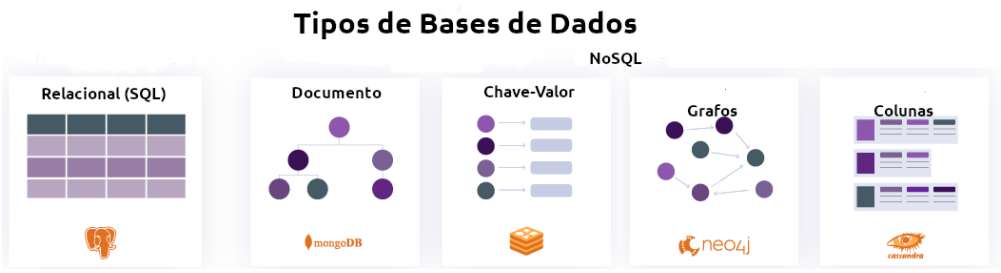

Figura 6: *Tipos de Bancos de Dados*

Sou um entusiasta do mundo **Open Source** e novas tecnologias. Qual a diferença entre Livre e Open Source? Livre significa que esta apostila é gratuita e pode ser compartilhada a vontade. Open Source além de livre todos os arquivos que permitem a geração desta (chamados de arquivos

fontes) devem ser disponibilizados para que qualquer pessoa possa modificar ao seu prazer, gerar novas, complementar ou fazer o que quiser. Os fontes da apostila (que foi produzida com o LaTex) está disponibilizado no GitHub [9]. Veja ainda outros artigos que publico sobre tecnologia através do meu Blog Oficial [7].

Referências

[1] Página do Redis
`https://redis.io`

[2] Página do Oracle Java
`http://www.oracle.com/technetwork/java`

[3] Página do Python
`https://www.python.org`

[4] Projeto GitHub do Another Redis Desktop Manager
`https://github.com/qishibo/AnotherRedisDesktopManager`

[5] Editor Spring Tool Suite para códigos Java
`https://spring.io/tools`

[6] Página do Jupyter
`https://jupyter.org/`

[7] Fernando Anselmo - Blog Oficial de Tecnologia
`http://www.fernandoanselmo.blogspot.com.br/`

[8] Encontre essa e outras publicações em
`https://cetrex.academia.edu/FernandoAnselmo`

[9] Repositório para os fontes da apostila
`https://github.com/fernandoans/publicacoes`