

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

FACULDADE DE CIÊNCIAS - CAMPUS BAURU

DEPARTAMENTO DE COMPUTAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DAVI AUGUSTO NEVES LEITE

GIOVANI CANDIDO

LUIS HENRIQUE MORELLI

**REDIS: ARMAZENAMENTO DE ESTRUTURA DE DADOS DE
CHAVE-VALOR NA MEMÓRIA**

BAURU

2022

DAVI AUGUSTO NEVES LEITE
GIOVANI CANDIDO
LUIS HENRIQUE MORELLI

REDIS: ARMAZENAMENTO DE ESTRUTURA DE DADOS DE CHAVE-VALOR NA MEMÓRIA

Trabalho da disciplina Banco de Dados II do
curso de Ciência da Computação da Univer-
sidade Estadual Paulista "Júlio de Mesquita
Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Aparecido Nilceu Marana

Resumo

O Redis é um banco de dados não relacional na memória que oferece alto desempenho e um modelo de dados exclusivo para produzir uma plataforma de solução de problemas. O objetivo principal do trabalho consiste, neste âmbito, reunir e apresentar as informações mais relevantes acerca do Redis, de tal forma que seja possível facilitar o entendimento do leitor diante do tema. Primeiramente, buscou-se artigos a respeito dos bancos de dados relacionais e não relacionais, por meio de periódicos como IEEE e CiteSeerX, além do uso de motores de busca como Google Scholar e Research Gate. Após isso, a documentação oficial foi consultada uma vez que possui uma seção somente para enumerar os livros extra-oficiais do Redis. Após isso, as informações coletadas foram filtradas e concatenadas, de tal forma que facilitasse o entendimento do leitor acerca das características, vantagens e desvantagens das abordagens relacional e não relacional de banco de dados, com foco no Redis. Por fim, tornou-se necessário a apresentação de um tutorial sobre a instalação e manuseio básico da ferramenta, bem como sua usabilidade com uma aplicação criada por meio da linguagem Python.

Palavras-chave: Banco de Dados Não Relacional; NoSQL; Redis

Lista de figuras

Figura 1 – Exemplo de tabela de um banco de dados relacional	6
Figura 2 – Incrementação da tabela "Empregado" com dados	7
Figura 3 – Exemplo de Consulta em SQL: seleção apenas dos nomes dos empregados que trabalham na cidade Kent	8
Figura 4 – Estrutura básica de um banco de dados NoSQL	9
Figura 5 – Tipos de um banco de dados NoSQL: Armazenamento de Chave-Valor, Orientados a Documentos e Orientados a Grafos.	10
Figura 6 – Instalação do Redis: adicionando repositório oficial	24
Figura 7 – Instalação do Redis: atualizando informações do repositório	24
Figura 8 – Instalação do Redis: instalando o NoSQL para Lubuntu	25
Figura 9 – Servidor Redis: abertura do servidor na máquina	25
Figura 10 – Execução do <i>redis-cli</i> : teste de conexão por meio do comando PING	26
Figura 11 – Execução do <i>redis-cli</i> : entrada para realização contínua de comandos . . .	27
Figura 12 – Execução do <i>redis-cli</i> : operações com o tipo <i>String</i>	28
Figura 13 – Execução do <i>redis-cli</i> : operações com o tipo <i>List</i>	28
Figura 14 – Execução do <i>redis-cli</i> : operações com o tipo <i>Sets</i>	29
Figura 15 – Execução do <i>redis-cli</i> : operações com o tipo <i>Hashes</i>	29
Figura 16 – Execução do <i>redis-cli</i> : operações com o tipo <i>Zset</i>	30
Figura 17 – Aplicação com Redis: instalação do conector para Python	31
Figura 18 – Aplicação com Redis: importação da biblioteca redis-py e criação da variável de conexão	31
Figura 19 – Aplicação com Redis: exemplo de manipulação no Python para cada tipo de dado do Redis	32
Figura 20 – Aplicação com Redis: resultado de execução do código	32

Lista de quadros

Quadro 1 – Comparativo de resultados em quatro diferentes bases de dados	34
--	----

Sumário

1	INTRODUÇÃO	6
1.1	Contexto	6
1.2	Justificativa	11
1.3	Objetivos	11
2	REVISÃO DE LITERATURA	12
2.1	Bancos de dados relacionais	12
2.2	Por que modelos não relacionais emergiram?	13
2.2.1	Incompatibilidade de impedância	14
2.2.2	Aplicações de banco de dados	14
2.2.3	Uso de <i>clusters</i> por grandes empresas da internet	15
2.2.4	Persistência poliglota	15
2.3	Bancos de dados não relacionais	15
2.3.1	Bancos de dados de chave-valor	16
2.3.2	Bancos de dados orientados a documentos	16
2.3.3	Bancos de dados em colunas	16
2.3.4	Bancos de dados baseados em grafos	17
2.4	Comparação entre modelos relacionais e não relacionais	17
2.5	Banco de dados Redis	18
2.5.1	História	19
2.5.2	Operação	19
2.5.3	Casos de uso	21
3	TUTORIAL	23
3.1	Instalação e Configuração	23
3.2	Exemplos Práticos	26
3.2.1	Linha de Comando	26
3.2.2	Aplicação em Python	30
4	METODOLOGIA	33
4.1	Coleta e Análise de Dados	33
5	CONSIDERAÇÕES FINAIS	36
	REFERÊNCIAS	37

Além disso, no modelo relacional existe um conceito básico para a identificação da linhas e estabelecimento de relações entre várias tabelas. Esse conceito é denominado chave que, resumidamente, representa um valor único que possa identificar cada uma das tuplas de uma tabela (HEUSER, 2009). O exemplo anterior do "Empregado", ao ser incrementado com dados, pode ser visto a seguir e é possível verificar a existência da chave para cada tupla.

Figura 2 – Incrementação da tabela "Empregado" com dados. Nota-se, essencialmente, que a chave é demonstrada pelo primeiro atributo, o qual representa um código único e que identifica cada tupla abaixo.

The diagram shows a table titled **EMPLOYEE** with three columns: **E#**, **Name**, and **Ort**. The data rows are as follows:

E#	Name	Ort
E19	Stewart	Stow
E4	Bell	Kent
E1	Murphy	Kent
E7	Howard	Cleveland

Annotations in the diagram:

- An arrow labeled "Column" points to the **Name** column header.
- A yellow box highlights the **Name** column, indicating it as the primary key.
- A yellow box highlights the entire row for **E4**, labeled "Record (row or tuple)".
- An arrow labeled "Data value" points to the value **Bell** in the **Name** column of the **E4** row.

Fonte: Meier e Kaufmann (2019)

Dessa forma, um banco de dados relacional essencialmente deve ofertar um conjunto de ferramentas para consulta e manipulação dos dados. Para tanto, existe a chamada *Structured Query Language* (SQL), ou Linguagem de Consulta Estruturada, criada especialmente para lidar com o modelo relacional (MEIER; KAUFMANN, 2019). Especificamente, como aborda Elmasri e Navathe (2011), o SQL foi um dos principais motivos para o grande sucesso dos bancos de dados relacionais, uma vez que facilitou a migração das aplicações entre os diferentes SGBD (Sistema de Gerenciamento de Banco de Dados) graças a sua padronização. Além disso, como adiciona Meier e Kaufmann (2019) em consonância com Elmasri e Navathe (2011), essa linguagem se baseia em um modelo declarativo de nível mais alto, ou seja, o usuário deve apenas especificar qual deve ser o resultado, tendo a otimização e outras decisões de gerenciamento de recursos à cargo do SGDB. Um exemplo pode ser visto na imagem a seguir, a qual representa uma consulta de seleção de dados, ou seja, o comando "SELECT-FROM-WHERE".

Figura 3 – Exemplo de Consulta em SQL: seleção apenas dos nomes dos empregados que trabalham na cidade Kent. Como pode ser visto, apenas três linhas foram necessárias para a seleção de dados.

EMPLOYEE		
E#	Name	City
E19	Stewart	Stow
E4	Bell	Kent
E1	Murphy	Kent
E7	Howard	Cleveland

Example query:
"Select the names of the employees living in Kent."

Formulation with SQL:

```
SELECT Name
FROM EMPLOYEE
WHERE City = 'Kent'
```

Results table:

Name
Bell
Murphy

Fonte: Meier e Kaufmann (2019)

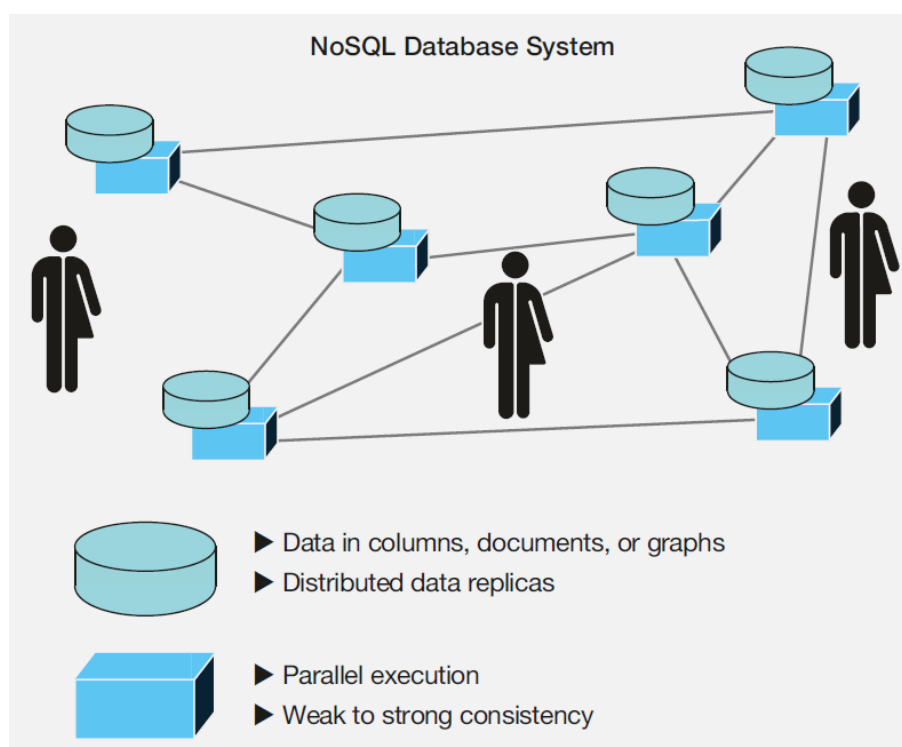
Diante disso, evidenciam-se algumas vantagens do modelo relacional e do uso do SQL, de acordo com Ali et al. (2019): baseada na estrutura pré-definida de ACID, isto é, Atomicidade, Consistência, Isolação e Durabilidade das informações; simplicidade de uso; facilidade em projetar, executar, manter e usar; informação é salva em apenas um lugar; melhora na integridade das informações; padronização na linguagem de consulta e segurança consolidada. Por outro lado, algumas desvantagens deste modelo são colocadas por Ali et al. (2019) e Khasawneh, AL-Sahlee e Safia (2020): na maioria dos SGBDs relacionais, há problemas na distribuição e armazenamento de grandes quantias de dados; questões relacionadas à flexibilidade e complexidade, no que se refere à necessidade de evidenciar uma estrutura predefinida dos dados; pode haver sobrecargas de *hardware* e dificuldade na recuperação de dados perdidos.

Em contrapartida e diante das desvantagens do banco de dados relacional, necessitou-se de uma nova solução que providenciasse um mecanismo de facilidade e velocidade no armazenamento e recuperação dos dados, independentemente de suas estruturas ou conteúdo. Para tanto, conforme coloca Sareen e Kumar (2015), os chamados bancos de dados não relacionais foram criados e, atualmente, compreendem grande uso por empresas como Google, Facebook e Amazon. A causa está ligada, principalmente, pela simplicidade de design, controle mais preciso sobre a disponibilidade dos dados e pela utilização de uma escala "horizontal" na distribuição de dados, tornando-o principal para aplicações que se baseiam no chamado *Big Data*. O *Big Data*, em poucas palavras, é um termo utilizado para se referir a existência de

uma grande quantia de dados digitais.

Dessa forma, uma definição para os bancos de dados não distribuídos está baseada na não exigência de altas relações entre os dados, junto a inexistência de um esquema pré-definido e, por consequência, é possível armazenar diversos tipos de dados (KHASAWNEH; AL-SAHLEE; SAFIA, 2020). Detalhadamente, Ali et al. (2019) explica que esse tipo de modelo, também chamado de NoSQL (*Not Only SQL* ou Não Apenas SQL), fornece um mecanismo para armazenar e recuperar dados independentemente de sua estrutura ou conteúdo, de tal modo que seja possível superar os problemas de restrições de cenários existentes no SQL e, diante disso, prover uma melhor comunicação entre servidores e clientes por meio de uma arquitetura distribuída. A imagem abaixo reflete a estrutura básica de um banco de dados NoSQL, por meio da distribuição de dados em execução paralela.

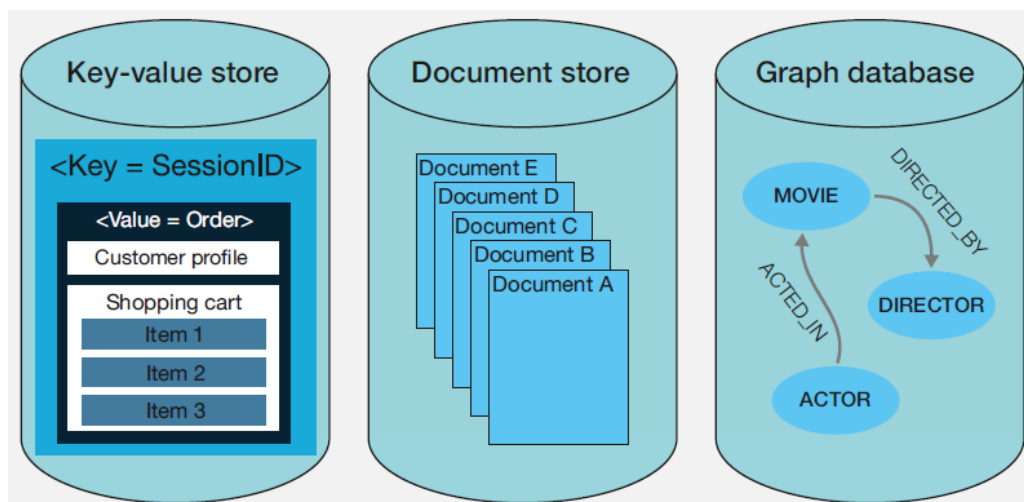
Figura 4 – Estrutura básica de um banco de dados NoSQL. Detalhadamente, é possível perceber que os dados são organizados em algum tipo de mecanismo e há replicação distribuída deles.



Fonte: Meier e Kaufmann (2019)

Com relação ao mecanismo (ou tipo) do banco de dados NoSQL, existem atualmente quatro categorias, de acordo com Sareen e Kumar (2015) e Khasawneh, AL-Sahlee e Safia (2020): Armazenamento de Chave-Valor; Orientados a Documentos; Orientados a Colunas e; Orientados a Grafos. Na imagem a seguir, é possível visualizar a estrutura dos tipos Armazenamento de Chave-Valor, Orientados a Documentos e Orientados a Grafos. Ressalva-se que os Orientados a Colunas representam uma variação do tipo Armazenamento de Chave-Valor e, portanto, devem ser considerados como parte do primeiro elemento da imagem.

Figura 5 – Tipos de um banco de dados NoSQL: Armazenamento de Chave-Valor, Orientados a Documentos e Orientados a Grafos.



Fonte: Meier e Kaufmann (2019)

Diante disso, algumas vantagens são elencadas por Khasawneh, AL-Sahlee e Safia (2020), Ali et al. (2019) e Sareen e Kumar (2015): não necessita de uma grande banda larga de dados para ser utilizado; pode rodar em dispositivos com baixas especificações de *hardware*; possui alta performance e baixa complexidade; maior flexibilidade e eficiência para aplicações de *Big Data*, uma vez que a escala é horizontal e não há restrições no modelo de dados; sem necessidade de administradores (DBMA) como no SQL; menor custo do que a abordagem relacional, uma vez que a maioria das ferramentas são de código aberto e não há grande necessidade de reparos ou de gerência. Contudo, Ali et al. (2019) e Nayak, Poriya e Poojary (2013) evidenciam algumas desvantagens do NoSQL: imaturidade e menor suporte das ferramentas, uma vez que possuem o surgimento mais recente do que as de SQL; alguns SGBDs não relacionais não suportam a metodologia ACID, complicando a migração; dificuldade na manutenção, uma vez que a metodologia é automatizada; sem uma interface específica, podendo dificultar o contato entre o banco de dados e certas aplicações.

Tendo em vista o exposto, o presente trabalho consiste em trazer informações e uma aplicabilidade a respeito do Redis que, como dito anteriormente, trata-se de um banco de dados não relacional remoto na memória e do tipo Armazenamento de Chave-Valor que oferece alto desempenho na aplicação. Ainda, como abordado por Macedo e Oliveira (2011), ele pode ser denominado de um servidor de estrutura de dados, uma vez que implementa estruturas de dados específicas para que as chaves contenham *strings* binárias seguras, *hashes*, *sets*, *sets* ordenados e listas, tornando essa ferramenta bem rápida e flexível.

Historicamente, o Redis teve seu projeto iniciado no início de 2009, por um italiano denominado Salvatore Sanfilippo. O principal motivo dessa criação, como aborda Macedo e Oliveira (2011) e Russo (2010), foi para melhorar a performance do chamado LLOOGG, um produto de Salvatore para análise web em tempo real. Com o tempo, o Redis obteve um

grande crescimento tanto na comunidade de código-aberto quanto no ramo empresarial, sendo utilizado atualmente em larga escala por grandes empresas como GitHub, Blizzard, Twitter, Tumblr, Pinterest, Instagram, The New York Times e diversas outras, conforme é colocado por Silva e Tavares (2015) e Macedo e Oliveira (2011).

Portanto, a construção deste trabalho torna-se importante uma vez que os objetivos incluem em reunir e apresentar as informações mais importantes a respeito dessa ferramenta e, com isso, pode se tornar uma referência importante para os novos desenvolvedores que desejam aprender e entender sobre esse banco de dados NoSQL. Por conta disso, o presente trabalho está estruturado em cinco principais capítulos. O primeiro capítulo traz uma breve explicação a respeito do banco de dados Redis, comentando e comparando sucintamente sobre as características, vantagens e desvantagens entre um banco de dados relacional e um não relacional, além dos objetivos e da justificativa da pesquisa. O segundo capítulo, por sua vez, está destinado ao levantamento das informações necessárias para compreensão do tema pelo leitor, ou seja, será detalhado a respeito dos bancos de dados relacionais e não relacionais, com foco exclusivo ao Redis. Após isso, o terceiro capítulo busca trazer dois exemplos práticos, realizado pelos autores, de tal modo que seja possível visualizar as características e comandos básicos do Redis em uma aplicação. Já o quarto capítulo demonstra as etapas de realização do trabalho, como a coleta e análise dos dados. Por fim, o quinto capítulo representa as considerações finais, ou seja, retoma a discussão inicial e verifica se os objetivos estabelecidos foram cumpridos.

1.2 Justificativa

O presente trabalho se justifica pela diversidade de informações acerca do tema e, desta maneira, há uma dificuldade dos desenvolvedores em filtrar os livros e artigos que tragam os dados mais importantes acerca do Redis. Diante disso, torna-se importante a criação de um material que reúna e apresente, de forma direta, as informações mais importantes a respeito da ferramenta, como o seu histórico e crescimento no mercado atual, suas vantagens e desvantagens com relação às abordagens tradicionais (como SQL) e usabilidade com relação às aplicações (exemplos práticos).

1.3 Objetivos

O objetivo principal deste trabalho, portanto, consiste em reunir e apresentar as informações mais importantes a respeito do Redis, um banco de dados não relacional. Especificamente, devem ser apresentados dois exemplos práticos, a fim de demonstrar a usabilidade da ferramenta para as aplicações.

2 Revisão de Literatura

Neste capítulo, introduz-se o conceito de bancos de dados e a sua relevância no século XXI, que abriga grandes empresas de internet. Assim, situa-se o leitor em relação aos modelos relacionais e aos não relacionais, evidenciando o motivo da emergência e os diferentes tipos de bancos não-relacionais. Ademais, procura-se discutir os prós e os contras de cada abordagem. Por fim, apresenta-se o Redis, objeto de estudo do presente trabalho, abordando a história de sua criação, evidenciando o seu modo de operação, além de descrever algumas de suas especificidades e apresentar as finalidades para quais Redis é frequentemente empregada.

Para uma melhor compreensão dos tópicos a serem expostos, faz-se necessário iniciar pelo conceito de banco de dados. Elmasri e Navathe (2011) definem banco de dados como uma coleção de dados relacionados que fazem referência a entidades do mundo real ou, em outras palavras, ao universo de discurso de um problema. Mais precisamente, os autores definem como uma coleção coerente de dados, que possuem um significado e que precisam ser armazenados devido a um propósito específico, haja vista que um grupo de pessoas e aplicações têm interesse nos dados. Outrossim, os autores exemplificam dados por meio de uma agenda telefônica, dizendo que dados são fatos conhecidos e que possuem algum significado, mesmo que implícito, como os nomes, telefones e endereços que uma pessoa mantém em sua agenda.

Elmasri e Navathe (2011) descrevem que algumas organizações precisam lidar com um grande volume de informações que precisam ser armazenadas e organizadas, permitindo que usuários recuperem tais informações e, se necessário, atualizem. Como exemplo de organização, pode-se citar a Amazon, que precisa armazenar as informações dos seus produtos, dos seus vendedores, dos seus compradores etc. Vale destacar que um banco de dados pode ser gerado manualmente, por meio de cadernetas por exemplo, sendo o avanço da computação o responsável pela criação dos bancos de dados computadorizados dos dias atuais.

2.1 Bancos de dados relacionais

Segundo Heuser (2009), um banco de dados baseado no modelo relacional é uma forma de armazenamento composta por tabelas, também denominadas relações. Ademais, uma tabela é constituída por linhas, também chamadas de tuplas, e cada linha contém um conjunto igual de campos, ou valores de atributos. Outrossim, os autores ainda descrevem que cada campo é identificado por um nome de campo, ou nome de atributo, e um conjunto de campos com mesmo nome forma uma coluna. Por fim, de acordo com Elmasri e Navathe (2011), o esquema de uma relação é dado pelo nome da relação e por sua lista de atributos.

Para mais, uma linha específica da tabela, em consonância com Heuser (2009), pode

ser identificada por uma chave única, a qual é denominada chave primária. A chave é constituída por um ou mais atributos, que formam uma combinação única. Ademais, podem haver chaves alternativas que também servem o propósito de identificar tuplas. Outro tipo de chave, conforme os autores, é a estrangeira. Esse tipo de chave simboliza um campo ou uma combinação de campos utilizados para estabelecer relacionamentos entre as tabelas, sendo situadas, por exemplo, em uma tabela x e identificando uma tabela y . Desta forma, pode-se afirmar que uma chave estrangeira em x é composta pelos campos da chave primária de y .

Em relação às tabelas, é necessário que cada uma deva armazenar atributos referentes a uma entidade ou a um relacionamento do universo de discurso (HEUSER, 2009). Para exemplificar, podemos pensar no contexto de uma escola. No modelo relacional, uma escola provavelmente teria as entidades Professor, Disciplina, Turma e Aluno. Alguns relacionamentos, com base nestas entidades, poderiam ser "Tem", "Contem" e "Ensina", pensando que a Disciplina "Tem" (tem uma ou mais) Turma, a Turma "Contem" (contém um ou mais) Aluno e o Professor "Ensina" (ensina uma ou mais) Turma. Nesse cenário, pode-se dizer que haveriam sete tabelas. No entanto, vale notar que o exemplo é simplista.

No que se refere aos atributos, cada um tem um domínio, que é um tipo primitivo padronizado e aceito pelo Sistema Gerenciador de Banco de Dados (SGBD). Ademais, os valores tem que ser atômicos, compostos por uma única unidade, e mono-valorados, dotados de um único valor primitivo. Tais restrições são gerenciadas pelo SGBD e diferem os bancos de dados das estruturas de dados que encontramos nas linguagens de programação, as quais podem ser mais complexas, como uma lista (HEUSER, 2009). É importante ressaltar que o SGBD é diferente para cada banco de dados relacional e trata-se da ferramenta responsável por gerenciar as relações, os relacionamentos, interpretar comandos etc.

Em termos de comandos, há uma única linguagem base para a definição e manipulação de dados, a SQL, do inglês *Structured Query Language*, ou Linguagem de Consulta Estruturada (ELMASRI; NAVATHE, 2011). Uma curiosidade interessante é que a linguagem anteriormente era chamada de SEQUEL, maneira como SQL é comumente pronunciada até os dias atuais. No mais, embora a linguagem seja comum aos bancos de dados relacionais, cada banco tem as suas peculiaridades mesmo quando se trata da linguagem SQL.

Dentre os bancos relacionais existentes, podemos citar alguns proprietários bem populares, como o Oracle e o Oracle Rdb, o DB2, o Informix Dynamic Server, o SQL/DS, o SQL Server, o SQL Access, o Ingress, o Sysbase e o mSQL. Por outro lado, tem-se as alternativas livres e de código aberto, como o MySQL, o PostgreSQL e o SQLite.

2.2 Por que modelos não relacionais emergiram?

Segundo Sadalage e Fowler (2013), a emergência dos bancos de dados não-relacionais, também denominados bancos NoSQL, foi motivada, em síntese, por três fatores: a incompa-

tibilidade de impedância, as aplicações de banco de dados e o uso de *clusters* por parte de grandes empresas de internet, como a Google e a Amazon. Ademais, um ponto interessante feito pelos autores é que o termo NoSQL foi, inicialmente, cunhado para designar um banco relacional de código aberto, o Strozzi NoSQL. Vale ainda notar que NoSQL é muito comumente interpretado como "Não SQL", quando, na verdade, bancos não-relacionais utilizam linguagens de consulta com funcionalidades muito próximas às do SQL ("Não Apenas SQL").

2.2.1 Incompatibilidade de impedância

Embora o modelo relacional tenha sido e ainda é um sucesso entre os desenvolvedores, há uma diferença entre o modelo e as estruturas de dados empregadas nos programas. Sadalage e Fowler (2013) afirmam que essa diferença é chamada de incompatibilidade de impedância. No modelo relacional, os dados são organizados em relações e tuplas, sendo que uma tupla é um conjunto de chaves e valores, enquanto uma relação é um conjunto de tuplas. Ademais, as operações desempenhadas sobre um banco são definidas formalmente pela álgebra linear.

Assim sendo, é evidente que a fundamentação matemática, além do esquema fixo das relações, limita os bancos de dados no que se refere a representação de estruturas mais complexas como as empregadas em memória. No modelo relacional, não é possível, por exemplo, armazenar uma lista como valor ou ter uma relação com tuplas de esquemas diferentes. Estruturas como listas precisam ser convertidas para o paradigma do modelo relacional. Outrossim, uma única estrutura agregada, como um objeto, que é bem capaz de representar entidades do mundo real, precisa ser decomposto em várias partes, cada uma armazenada em uma relação.

Nesse contexto, como Sadalage e Fowler (2013) comentam, vale lembrar que surgiram alguns bancos de dados orientados a objetos que tentaram superar a incompatibilidade supracitada. No entanto, como o modelo relacional era e é um ponto forte de ligação entre os desenvolvedores e os administradores de bancos, esse novo modelo não foi bem sucedido.

2.2.2 Aplicações de banco de dados

Faz-se necessário pontuar, em concordância com Sadalage e Fowler (2013), que a popularidade dos bancos relacionais pode ter sido impulsionada pela integração que tais bancos proporcionavam e ainda proporcionam. É bem comum que uma base de dados seja acessada por diversas aplicações, que muitas vezes são criadas por diferentes times de desenvolvimento.

Desse modo, o esquema fixo das tabelas e os tipos de dados padrões permitem que as diversas aplicações armazenem dados em uma única base de dados que tem uma interface conhecida por todos os times. No entanto, quando se trata de aplicações mantidas por um único time e que fazem uso de bancos de dados, não há necessidade de uma interface comum, sendo possível que o time mantenha um banco sem esquema definido para cada entidade, que possa armazenar uma estrutura de dados complexa etc. Em outras palavras, o time pode optar

por uma alternativa mais flexível e que se adapte melhor às necessidades da aplicação.

2.2.3 Uso de *clusters* por grandes empresas da internet

Ainda segundo Sadalage e Fowler (2013), o modelo relacional não acompanhou bem a internet do século do XXI, uma vez que as propriedades web cresceram consideravelmente e muitas empresas precisaram lidar com grandes quantidades de dados, o que necessitava de um grande poder de processamento de dados. O fato não é que o modelo relacional não foi capaz de lidar bem com os novos requerimentos, já que, para processar esta nova demanda, as grandes empresas passaram a manter *clusters*, um conjunto de computadores que deveriam funcionar como um só grande e potente computador. Além de não serem projetados para trabalhar em *clusters*, os bancos relacionais, que eram licenciados com a premissa de que um único servidor seria empregado, aumentaram os preços em decorrência dos *clusters*.

2.2.4 Persistência poliglota

Com o crescimento do volume de dados que as empresas de internet precisaram lidar e a inconciliabilidade do modelo relacional com os *clusters*, as grandes empresas passaram a buscar alternativas para o armazenamento de dados. Google e Amazon tomaram a frente e criaram, respectivamente, o BigTable e o Dynamo, dois bancos não-relacionais para uso interno (SADALAGE; FOWLER, 2013). Com o tempo, outras empresas passavam pela mesma dificuldade, o que levou ao desenvolvimento de mais bancos de dados não-relacionais. Vale lembrar que, por causa da questão do licenciamento dos bancos relacionais para *clusters*, muitas conferências foram realizadas para a discussão de projetos NoSQL de código aberto.

Sadalage e Fowler (2013) afirmam que diversos bancos não-relacionais de código aberto passaram a integrar a lista de opções para o armazenamento de dados por parte das empresas. Ademais, a Google e a Amazon liberaram versões públicas dos bancos BigTable e Dynamo que criaram para uso interno. Outrossim, os bancos relacionais passaram a ser vistos pelos desenvolvedores como uma opção, o que deu origem ao termo persistência poliglota.

O termo persistência poliglota se refere às diferentes escolhas de armazenamento que os desenvolvedores passaram a ter. Ao invés de utilizarem bancos relacionais por ser o mais comum até então, passou-se a empregar diferentes bancos, visando implementar o mais adequado para o projeto. Neste sentido, grandes empresas puderam concatenar múltiplas tecnologias de armazenamento em múltiplas situações distintas (SADALAGE; FOWLER, 2013).

2.3 Bancos de dados não relacionais

Como evidenciado na sessão 2.2, os bancos não relacionais surgiram como alternativas ao modelo relacional, oferecendo recursos específicos para certos tipos de uso, flexibilidade,

suporte apropriado aos *clusters* e uma licença gratuita proveniente da filosofia de código aberto. Vale ressaltar que os recursos peculiares podem ser evidenciados pelos diferentes modelos não-relacionais. De acordo com Sadalage e Fowler (2013), os bancos não-relacionais, ou NoSQL, podem ser classificados como: Bancos de dados de chave-valor, Bancos de dados orientados a documentos, Bancos de dados em colunas e Bancos de dados baseados em grafos.

Os diferentes tipos de bancos não relacionais são apresentados nas subseções 2.3.1, 2.3.2, 2.3.3 e 2.3.4. Vale ressaltar as explicações foram extraídas de Sadalage e Fowler (2013).

2.3.1 Bancos de dados de chave-valor

Uma estrutura de armazenamento de chave-valor é basicamente uma tabela de dispersão, também conhecida como tabela *hash*. Nesta tabela, o acesso a um determinado valor é feito por meio de uma chave única, que o identifica. Deste modo, podemos entender um banco chave-valor como um banco relacional de apenas duas colunas, no qual uma coluna é uma chave primária, como um campo ID, e a outra coluna é um valor, como um campo nome.

Dentre os bancos de chave-valor existentes, podemos citar alguns bem populares, como o Riak, o Redis, o Memcached, o Berkeley DB, HamsterDB, Amazon DynamoDB e o Project Voldemort. Vale destacar que a única opção de código fechado é o Amazon DynamoDB, que deu origem ao Voldemort, uma alternativa de código aberto.

2.3.2 Bancos de dados orientados a documentos

Os bancos de dados orientados a documentos empregam uma estrutura chave-valor semelhante aos armazenamentos de chave-valor. No entanto, ao invés de um valor, é armazenado um documento na tabela *hash*. Estes documentos podem ser dos tipos XML, JSON, BSON, entre outros. Tais documentos organizam as informações de acordo com a estrutura imposta pelo projetista e não é preciso que todos possuam os mesmos atributos, mesmo que pertençam a uma mesma coleção de valores. Os atributos são definidos dinamicamente, cabendo ao projetista manipular adequadamente cada documento.

A lista dos bancos orientados a documentos que se tornaram bem populares inclui o MongoDB, o CouchDB, o Terrastore, o OrientDB, o RavenDB e o Lotus Notes.

2.3.3 Bancos de dados em colunas

O armazenamento colunar também emprega uma estrutura chave-valor. Porém, cada chave endereça uma família de colunas. Em outras palavras, cada chave dá acesso a um conjunto de dados relacionados. Tais dados são endereçados pela mesma chave, tendo em mente que são valores geralmente acessados ao mesmo tempo. Como exemplo, podemos citar as informações pessoais de um cliente no sistema de uma loja de e-commerce.

Ademais, as ferramentas mais comuns de armazenamento colunar são o Apache Cassandra, o Apache HBase, o Hypertable e o Amazon DynamoDB.

2.3.4 Bancos de dados baseados em grafos

O armazenamento com base em grafos permite guardar entidades e estabelecer relacionamentos entre eles, por meio de grafos. Neste tipo de banco, cada nó representa uma entidade, que pode ter atributos. Cada aresta representa um relacionamento entre duas entidades, a qual possui atributos e tem diferente significado dependendo da direção. Desse modo, percorrer o grafo permite que os dados sejam facilmente interpretados. Como exemplo, podemos pensar que cada entidade é uma pessoa, que tem um atributo nome, e uma aresta "trabalha para" entre duas pessoas pode indicar um relacionamento entre as duas.

Dentre as escolhas mais comuns para um banco de dados baseado em grafos, tem-se o Neo4J, o Infinite Graph, o OrientDB e o FlockDB.

2.4 Comparação entre modelos relacionais e não relacionais

Ali et al. (2019) realizaram uma comparação entre os bancos relacionais e os bancos NoSQL, catalogando vantagens e desvantagens do NoSQL em relação aos relacionais, além de prós e contras dos relacionais em relação aos NoSQL. Com base no estudo dos autores, os parágrafos seguintes apresentam os principais achados em relação ao desempenho dos modelos.

No que se refere aos prós dos bancos não relacionais em relação aos relacionais, pode-se citar que os bancos NoSQL disponibilizam uma série de modelos de dados distintos, como o modelo chave-valor, o modelo orientado a documentos, o modelos orientado a colunas e o modelo baseado em grafos; tais bancos também são escaláveis mais facilmente; ainda, eles não requerem um administrador de banco de dados; podem lidar com falhas de hardware automaticamente; são mais flexíveis em relação aos tipos de dados que se podem armazenar e considerando que não possuem um esquema fixo para cada entidade ou relacionamento que possam armazenar; são mais eficientes e rápidos; foram desenvolvidos muito rapidamente pela comunidade e tem apoio de uma ampla gama de desenvolvedores, uma vez que a maior parte é de código aberto; são adequados para a execução em *clusters* e aplicações de *Big Data*, já que são menos custosos, sendo que a filosofia do código aberto fez com que fossem também livres para qualquer tipo de uso sem licenciamento; por fim, pode-se dizer que os bancos não-relacionais, com sua variedade de modelos e SGBDs, são empregados com o propósito geral de armazenar dados, mas possuem peculiaridades e casos de usos mais específicos, como explanado na seção 2.3 e como será explicado na seção 2.5.

Por outro lado, os bancos NoSQL possuem alguns inconvenientes, como a falta de maturidade, uma vez que são modelos recentes e que foram rapidamente desenvolvidos; tem-se também a falta de uma linguagem padrão para definição e consulta dos dados, embora as

linguagens dos bancos não-relacionais tentem ser similares ao SQL para facilitar o aprendizado dos desenvolvedores; além do mais, os bancos não-relacionais não tem *compliance* com a estrutura ACID, uma vez que não garante que as propriedades sejam respeitadas; não têm uma interface comum aos desenvolvedores; é mais difícil de ser mantido; possuem menos respaldo, uma vez que, muitos projetos são livres e mantidos pela comunidade, podendo não haver uma organização que ofereça suporte aos adeptos da tecnologia; por fim, pode-se dizer que os bancos relacionais são empregados somente com o propósito geral de armazenar dados.

No que se trata das vantagens dos bancos relacionais em relação aos bancos NoSQL, pode-se listar que o modelo relacional é mais simples de empregar, além de ser mais simples de projetar, operar e realizar manutenção; eles oferecem uma série de interfaces comuns aos desenvolvedores; garantem a integridade dos dados armazenados; oferecem uma linguagem padrão de definição e de consulta dos dados. Por fim, no que se refere aos inconvenientes do modelo relacional em relação ao NoSQL, pode-se citar o custo de licenciamento do banco de dados; a sobrecarga de hardware; as limitações estruturais das relações; ainda, dados perdidos são dificilmente recuperáveis nos bancos relacionais; os bancos relacionais não operam bem em *clusters* ou em aplicações de *Big Data*; por fim, algumas aplicações podem ter o processamento impactado negativamente pelo uso de bancos relacionais.

2.5 Banco de dados Redis

Segundo Silva e Tavares (2015), o Redis é um armazenamento de estrutura de dados, também denominado *datastore*, de chave-valor, na memória e de código aberto. O autor conta que Redis é a forma contraída de Remote Dictionary Server, que em português significa Servidor de Dicionário Remoto. Ademais, essa tecnologia NoSQL foi criada pelo italiano Salvatore Sanfilippo, em 2009, com a linguagem de programação C. Conforme Russo (2010) complementa, Sanfilippo deu início ao projeto, que hoje é mantido por muitos contribuidores e patrocinado pelo Redis Labs¹, visando otimizar o desempenho de um produto da sua startup.

Silva e Tavares (2015) descrevem que Redis tem clientes para mais de 30 linguagens de programação, sendo um dos bancos NoSQL mais populares entre os desenvolvedores e as grandes empresas de internet. Os autores afirmam que, dentre as grandes organizações que empregam o Redis, tem-se nomes como o Twitter, o GitHub, o Tumblr, o Pinterest, o Instagram, o Hulu, o Flickr e o The New York Times. Vale ressaltar que o código fonte do Redis está disponível no GitHub², além de ter uma documentação³ pública bem completa.

¹ <https://redis.com/>

² <https://github.com/antirez/redis>

³ <https://redis.io/documentation>

2.5.1 História

No que se refere a história do Redis, Macedo e Oliveira (2011) e Russo (2010) descrevem que essa tecnologia de armazenamento foi criada pelo desenvolvedor de software italiano Salvatore Sanfilippo no início de 2009. Os autores contam que a motivação do projeto era otimizar a performance de um produto da startup de Sanfilippo, que foi o LLOOGG, tendo sido criado especialmente para um processo de análise de dados de internet em tempo real denominado Web Analytics. Sendo assim, nota-se que esse banco de dados NoSQL surgiu por uma necessidade de desempenho e velocidade no tratamento de uma grande quantia dados.

Após a sua criação, a tecnologia cresceu rapidamente em termos de popularidade, recebendo grande suporte da comunidade de código aberto, que adicionou funcionalidades em passos largos. Ademais, tendo em vista o exposto, a empresa VMWare, conhecida pela produção de software e pela oferta de serviços de computação em nuvem e de virtualização, sendo também uma das principais contribuidoras do Redis, passou a trabalhar oficialmente no desenvolvimento do Redis, uma vez que Sanfilippo fechou um acordo com a empresa em 2010.

Por fim, é importante comentar sobre dois acontecimentos bem relevantes e atuais relacionados ao Redis. De acordo com Sanfilippo ([entre 2012 e 2021]), houve a criação de uma organização que ficou encarregada de patrocinar e fomentar o desenvolvimento e a manutenção da ferramenta de armazenamento em memória; isto aconteceu, mais especificamente, em junho de 2015 e a organização em questão foi nomeada Redis Labs. O segundo acontecimento, em consonância com o autor, foi quando ele deixou o projeto, em junho de 2020.

2.5.2 Operação

Carlson (2013) afirma que o Redis é uma base de dados remota, o que não necessariamente significa que os dados são armazenados em uma segunda máquina que fica em um local distinto. Na verdade, Redis é um software que emprega uma arquitetura cliente-servidor, mas tanto o servidor quanto o cliente podem ser executados na mesma máquina. A ideia é que um servidor esteja sempre executando na máquina, permitindo que os desenvolvedores possam, por meio de uma linguagem de programação de preferência, criar uma aplicações cliente que possa armazenar e consumir os dados por meio de requisições ao servidor.

Ademais, conforme supracitado, Redis faz uso de um lógica de chave-valor. Deste modo, o armazenamento de dados funciona como uma tabela *hash*, que é uma tabela de mapeamento de chave-valor na qual há várias entradas, cada uma sendo acessada pela chave e retornando um valor respectivo, em consonância com a explicação da subseção 2.3.1. Outrossim, de acordo com Silva e Tavares (2015), Redis suporta tipos de dados e estrutura de dados como *Strings*, *Hashes*, Listas, *Sets*, *Sets Ordenados*, *Bitmaps* e *HyperLogLogs*.

Redis é também uma forma de armazenamento na memória, o que significa que os dados ficam na memória principal da máquina, possibilitando que milhões de requisições sejam

feitas por segundo, uma vez que leituras e escritas se tornam muito mais rápidas quando comparadas às instruções de um armazenamento no disco (CARLSON, 2013). Ademais, o banco de dados é capaz de escalar e lidar desde um protótipo de uma aplicação até uma aplicação robusta em produção que precisa armazenar ou consumir centenas de *gigabytes* de dados.

É claro que, conforme o volume de dados aumenta, a memória principal da máquina pode ficar sobrecarregada, uma vez que a capacidade, em termos de gigabytes, é inferior a capacidade de discos rígidos, cujo destaque está associado, justamente, a quantidade que se pode armazenar ao invés da velocidade de leitura ou de escrita. Sendo assim, segundo Labs (2022), o Redis implementa um mecanismo semelhante ao de paginação de memória que os sistemas operacionais utilizam. O Redis, quando necessário, mantém todas as chaves na memória, porém somente os valores mais frequentemente acessados são mantidos na memória, os demais valores são mantidos em memórias flash, como as Unidades de Estado Sólido, ou *Solid State Drive* (SSD), que possuem maior capacidade do que as memórias principais e maior velocidade do que um disco rígido. Tal mecanismo é chamado de Redis on Flash (RoF).

Embora o Redis seja um *datastore* na memória, é possível, conforme abordado por Silva e Tavares (2015) e Carlson (2013), persistir os dados em dispositivo de armazenamento não volátil, como um disco rígido ou uma unidade de estado sólido. Segundo Silva e Tavares (2015), a persistência dos dados em Redis é dada por meio de *snapshots* ou de *journaling*. Em outras palavras, o Redis pode ser programado para criar cópias binárias instantâneas dos dados armazenados em memória, que são os *snapshots*, ou pode salvar em um arquivo toda a sequência de comandos executados sobre a base de dados em um determinado intervalo de tempo, possibilitando que a sequência seja executada novamente. Outrossim, a persistência bem como a replicação e fragmentação dos dados permitem que o Redis seja escalável horizontalmente, isto é, permitem que o Redis seja facilmente distribuído entre diversos nós.

Carlson (2013) afirma que, embora os bancos não-relacionais não possuam uma linguagem padrão como a SQL, o Redis possui comandos para manipulação e consulta de dados. Outrossim, de acordo com Silva e Tavares (2015), o Redis permite a criação de comandos adicionais por meio da linguagem de programação Lua. Vale ressaltar que os comandos possuem uma sintaxe comum aos desenvolvedores, empregando terminologias populares em linguagens orientadas a objetos e associadas a estruturas de dados específicas.

Ainda em relação aos comandos, Silva e Tavares (2015) contam que a ferramenta oferece suporte ao padrão Pub/Sub, que significa *Publish-Subscribe* ou Publicar-Assinar. Tal funcionalidade permite que um cliente envie mensagens a um certo canal e que todos os assinantes desse canal recebam a mensagem. Ademais, o autor expressa que o Redis suporta o *pipeline*, que é uma forma de enviar múltiplos comandos simultaneamente ao servidor Redis sem necessitar esperar por respostas individuais, lidas todas de uma única vez pelo cliente.

Para mais, Redis suporta transações, as quais permitem a execução serial de uma sequência de comandos como uma operação atômica. Vale destacar que, diferentemente dos

relacionais, o Redis só serve a um cliente durante a execução da transação e, caso ocorra uma falha, a execução é resumida a partir do comando em sequência (SILVA; TAVARES, 2015).

2.5.3 Casos de uso

No que se refere aos casos de uso do Redis, é possível afirmar que essa ferramenta pode ser empregado em diversas situações; porém, de acordo com Services (2022), pode-se citar que as principais aplicações dessa tecnologia são: a armazenagem em cache; a criação de chats, sistemas de mensagens e filas; placares de jogos; armazenamento de sessões; *streaming* de mídia avançada; dados geoespaciais; aprendizado de máquina; e análise em tempo real. Nos parágrafos a seguir, cada um dos usos é brevemente comentado com base a explicação da Services (2022) e dos conceitos anteriormente apresentados no presente trabalho.

Conforme o explanado anteriormente, o Redis é um *datastore* em memória, logo todos os dados residem na memória principal da máquina, possibilitando um acesso muito mais veloz do que o acesso aos dados armazenados em um banco de dados em disco. Assim sendo, pode-se usar o Redis como um mecanismo de cache para uma base de dados principal que fica armazenado em disco. Desta forma, ao invés de buscar dados diretamente no disco, que tem um tempo de acesso consideravelmente inferior ao da memória, uma aplicação pode consultar primeiro a porção dos dados que é mantida pelo Redis. É claro que, nesse contexto, o Redis armazenaria os dados requeridos mais recentemente ou com maior frequência.

Sendo uma forma de armazenamento em memória, o Redis é adequado para o armazenamento de dados referentes a sessão de um usuário em uma determinada aplicação web. Um exemplo bem simples pode ser a opção que um usuário faz por um tema escuro ou por um tema claro, uma vez que é necessário armazenar essa informação para que durante a navegação e o uso do usuário, o tema escolhido seja preservado. É claro que a informação pode ser persistida para que entre uma sessão e outra a opção também seja mantida.

Ademais, o rápido acesso aos dados armazenado no banco em decorrência do *datastore* de memória permite o Redis armazenar metadados sobre perfis de usuários, informações e *tokens* de autenticação utilizados por eles e visualização de históricos, bem como armazenamento de arquivos manifesto, possibilitando que CDNs (Rede de Entrega de Conteúdo) façam *streaming* de vídeo para milhões de usuário, estejam eles conectados em seus dispositivos.

Outrossim, além de possibilitar a utilização do Redis como armazenamento em cache e *streaming* de mídia ao vivo, a agilidade do *datastore* do Redis permite a criação, treinamento e implementação rápida de modelos de aprendizado de máquina, além do consumo, processamento e análise de dados em tempo real com baixas latências.

Por contar com o padrão Pub-Sub, o Redis é ideal para o envio de mensagens por meio de um cliente a um canal com um ou mais clientes. Desta forma, é viável a criação de chats ou sistemas e filas de mensagens. Para mais, a combinação do padrão Pub-Sub com a

estrutura de dados *sorted set*, um conjunto chave-valor ordenado por valor, o Redis viabiliza a criação e a transmissão de placares de jogos em tempo real.

O Redis possui estruturas e operadores de dados específicos em sua memória que permitem armazenar, processar e analisar dados geoespaciais, permitindo adicionar aos aplicativos recursos baseados em localização, tais quais tempo de percurso, distância e pontos de interesse, em tempo real e de maneira eficiente, para aplicações de grande escala.

3 Tutorial

Este capítulo tem como objetivo apresentar um guia para a instalação e configuração inicial do Redis. Também, para fins didáticos, serão apresentados dois exemplos práticos, sendo o primeiro baseado em linhas de comando de um terminal e o segundo baseado em uma aplicação escrita na linguagem Python. Dessa forma, será possível visualizar tanto a utilização direta da ferramenta quanto de forma indireta, isto é, por meio de um conector para uma linguagem de programação. Por fim, é importante mencionar que os seguintes materiais, além da documentação oficial dada por Labs (2022), foram utilizados para estabelecer todo o passo a passo: Carlson (2013) e Silva e Tavares (2015) para a instalação, configuração e primeiro exemplo prático; e Macedo e Oliveira (2011) para o segundo exemplo prático. Cada passo dos tutoriais possuem imagens representativas, de tal maneira a facilitar o leitor.

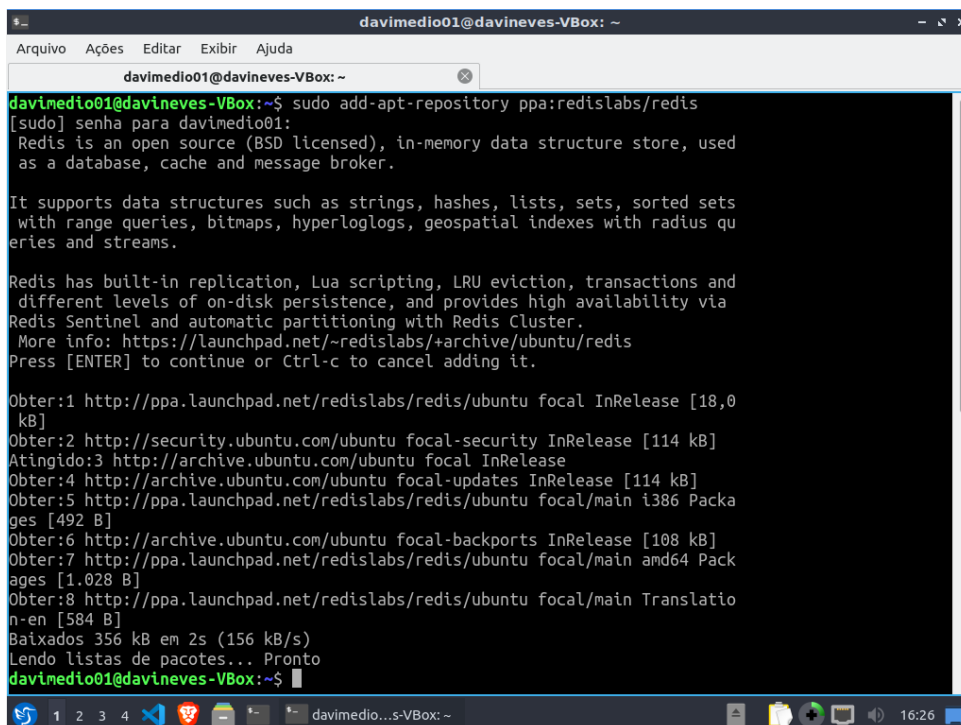
3.1 Instalação e Configuração

Antes de tudo, como ressalta Silva e Tavares (2015), o Redis possui suporte **somente** para sistemas baseados em Unix, isto é, Ubuntu, CentOS, Debian e outros semelhantes. Além disso, é necessário a instalação do pacote *build-essentials* para prover, sobretudo, diversas ferramentas para que seja possível a execução da ferramenta (como *gcc*, *make* e outras). Desta forma, o tutorial está baseado na utilização da versão 20.04 LTS do sistema Ubuntu, uma variação leve, rápida e segura do Ubuntu.

De início, é necessário abrir o terminal e executar os seguintes comandos em ordem:

1. "sudo add-apt-repository ppa:redislabs/redis": adicionar o repositório oficial do Redis nos gerenciador de pacotes do sistema;

Figura 6 – Instalação do Redis: adicionando repositório oficial.



```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda
davimedio01@davineves-VBox: ~
davimedio01@davineves-VBox:~$ sudo add-apt-repository ppa:redislabs/redis
[sudo] senha para davimedio01:
Redis is an open source (BSD licensed), in-memory data structure store, used
as a database, cache and message broker.

It supports data structures such as strings, hashes, lists, sets, sorted sets
with range queries, bitmaps, hyperloglogs, geospatial indexes with radius qu
eries and streams.

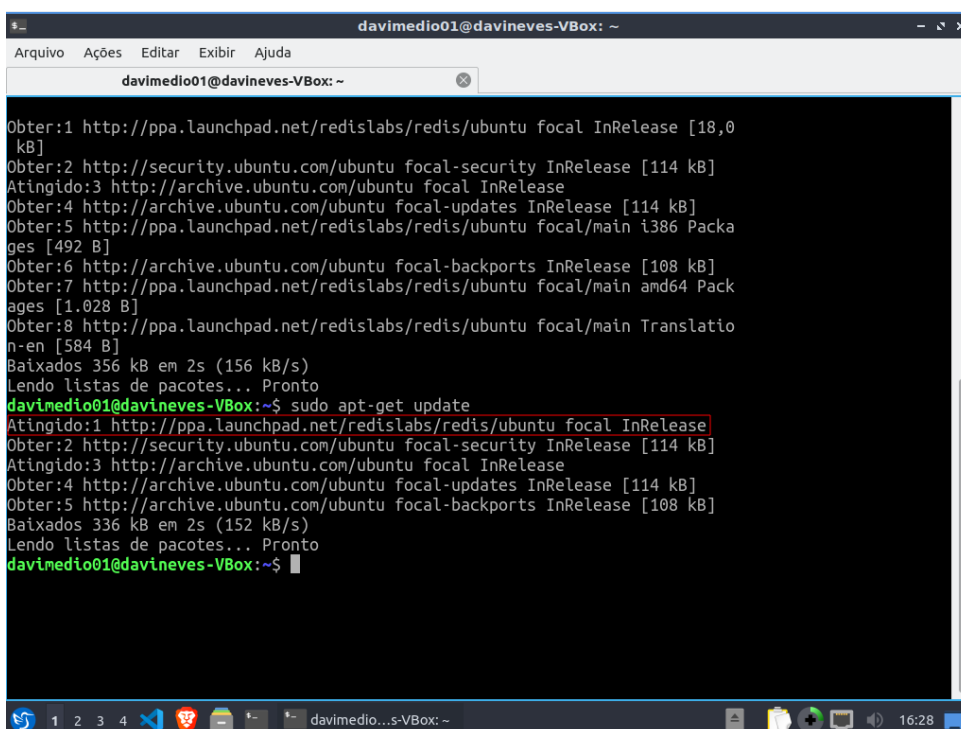
Redis has built-in replication, Lua scripting, LRU eviction, transactions and
different levels of on-disk persistence, and provides high availability via
Redis Sentinel and automatic partitioning with Redis Cluster.
More info: https://launchpad.net/~redislabs/+archive/ubuntu/redis
Press [ENTER] to continue or Ctrl-c to cancel adding it.

Obter:1 http://ppa.launchpad.net/redislabs/redis/ubuntu focal InRelease [18,0
kB]
Obter:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Atingido:3 http://archive.ubuntu.com/ubuntu focal InRelease
Obter:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Obter:5 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main i386 Packa
ges [492 B]
Obter:6 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Obter:7 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main amd64 Pack
ages [1.028 B]
Obter:8 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main Translatio
n-en [584 B]
Baixados 356 kB em 2s (156 kB/s)
Lendo listas de pacotes... Pronto
davimedio01@davineves-VBox:~$
```

Fonte: elaborado pelos autores, 2022.

2. "sudo apt-get update": realizar uma atualização relacionada a todos os repositórios existentes no sistema, de tal modo a coletar as últimas informações disponíveis;

Figura 7 – Instalação do Redis: atualizando informações do repositório.

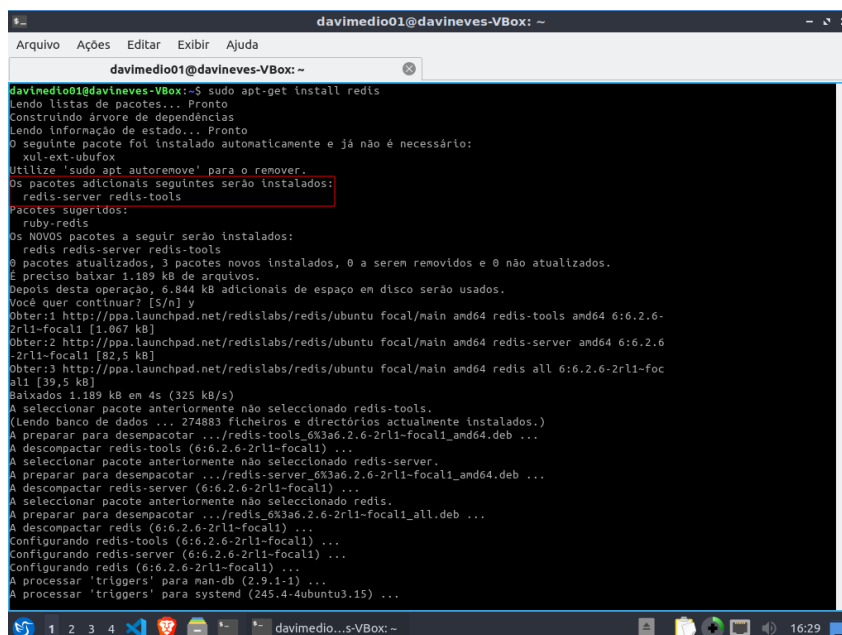


```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda
davimedio01@davineves-VBox: ~
Obter:1 http://ppa.launchpad.net/redislabs/redis/ubuntu focal InRelease [18,0
kB]
Obter:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Atingido:3 http://archive.ubuntu.com/ubuntu focal InRelease
Obter:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Obter:5 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main i386 Packa
ges [492 B]
Obter:6 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Obter:7 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main amd64 Pack
ages [1.028 B]
Obter:8 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main Translatio
n-en [584 B]
Baixados 356 kB em 2s (156 kB/s)
Lendo listas de pacotes... Pronto
davimedio01@davineves-VBox:~$ sudo apt-get update
Atingido:1 http://ppa.launchpad.net/redislabs/redis/ubuntu focal InRelease
Obter:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Atingido:3 http://archive.ubuntu.com/ubuntu focal InRelease
Obter:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Obter:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Baixados 336 kB em 2s (152 kB/s)
Lendo listas de pacotes... Pronto
davimedio01@davineves-VBox:~$
```

Fonte: elaborado pelos autores, 2022.

3. "sudo apt-get install redis": instalar o Redis a partir do repositório oficial para Ubuntu.

Figura 8 – Instalação do Redis: instalando o NoSQL para Lubuntu.



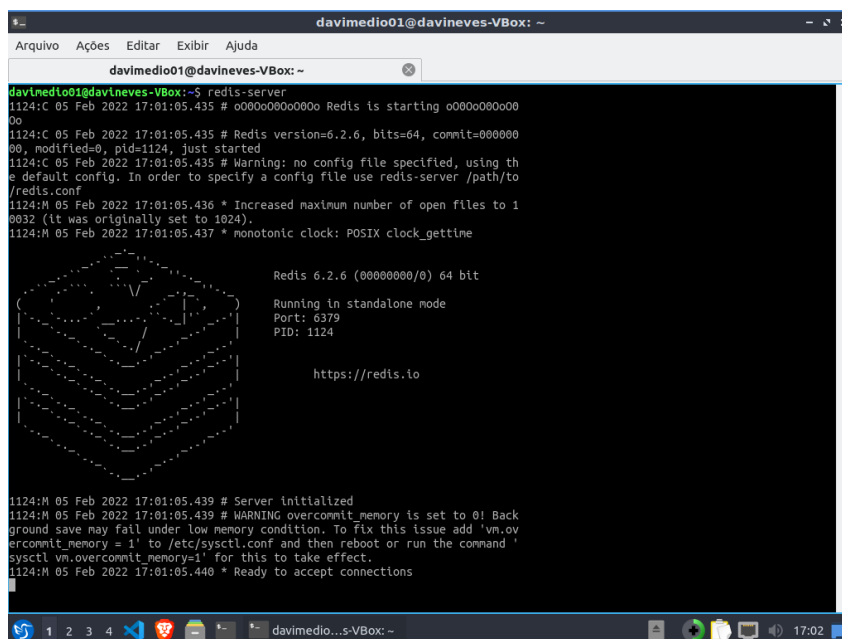
```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda

davimedio01@davineves-VBox: ~
davimedio01@davineves-VBox:~$ sudo apt-get install redis
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
O seguinte pacote foi instalado automaticamente e já não é necessário:
  xul-ext-ubufex
Utilize 'sudo apt autoremove' para o remover.
Os pacotes adicionais seguintes serão instalados:
  redis-server redis-tools
Pacotes supérfluos:
  ruby-redis
Os NOVOS pacotes a seguir serão instalados:
  redis redis-server redis-tools
O pacote atualizado, 3 pacotes novos instalados, 0 a serem removidos e 0 não atualizados.
É preciso baixar 1.189 kB de arquivos.
Depois desta operação, 6.844 kB adicionais de espaço em disco serão usados.
Você quer continuar? [S/n] y
Obter:1 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main amd64 redis-tools amd64 6:6.2.6-2r1-focal1 [1.067 kB]
Obter:2 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main amd64 redis-server amd64 6:6.2.6-2r1-focal1 [92,5 kB]
Obter:3 http://ppa.launchpad.net/redislabs/redis/ubuntu focal/main amd64 redis all 6:6.2.6-2r1-focal1 [39,5 kB]
Baixados 1.189 kB em 4s (325 kB/s)
A selecionar pacote anteriormente não selecionado redis-tools.
(Lendo banco de dados ... 274883 ficheiros e directorios actualmente instalados.)
A preparar para descompactar .../redis-tools_6x3a6.2.6-2r1-focal1_amd64.deb ...
A descompactar redis-tools (6:6.2.6-2r1-focal1) ...
A selecionar pacote anteriormente não selecionado redis-server.
A preparar para descompactar .../redis-server_6x3a6.2.6-2r1-focal1_amd64.deb ...
A descompactar redis-server (6:6.2.6-2r1-focal1) ...
A selecionar pacote anteriormente não selecionado redis.
A preparar para descompactar .../redis_6x3a6.2.6-2r1-focal1_all.deb ...
A descompactar redis (6:6.2.6-2r1-focal1) ...
Configurando redis-tools (6:6.2.6-2r1-focal1) ...
Configurando redis-server (6:6.2.6-2r1-focal1) ...
Configurando redis (6:6.2.6-2r1-focal1) ...
A processar 'triggers' para man-db (2.9.1-1) ...
A processar 'triggers' para systemd (245.4-4ubuntu3.15) ...
```

Fonte: elaborado pelos autores, 2022.

Com isso, o passo de instalação está concluído e o Redis está instalado no sistema. Uma vez instalado, é possível executar todas as ferramentas existentes, como o *redis-server*, que é o servidor do Redis. Desta forma, ao executar o comando, a seguinte imagem é exibida.

Figura 9 – Servidor Redis: abertura do servidor na máquina.



```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda

davimedio01@davineves-VBox: ~
davimedio01@davineves-VBox:~$ redis-server
1124:C 05 Feb 2022 17:01:05.435 # 000000000000 Redis is starting 0000000000
00
1124:C 05 Feb 2022 17:01:05.435 # Redis version=6.2.6, bits=64, commit=00000000
00, modified=0, pid=1124, just started
1124:C 05 Feb 2022 17:01:05.435 # Warning: no config file specified, using the
default config. In order to specify a config file use redis-server /path/to
/redis.conf
1124:M 05 Feb 2022 17:01:05.436 * Increased maximum number of open files to 1
0032 (it was originally set to 1024).
1124:M 05 Feb 2022 17:01:05.437 * monotonic clock: POSIX clock_gettime

Redis 6.2.6 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 1124

https://redis.io

1124:M 05 Feb 2022 17:01:05.439 # Server initialized
1124:M 05 Feb 2022 17:01:05.439 # WARNING overcommit_memory is set to 0! Back
ground save may fail under low memory condition. To fix this issue add 'vm.o
vercommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command '
sysctl vm.overcommit_memory=1' for this to take effect.
1124:M 05 Feb 2022 17:01:05.440 * Ready to accept connections
```

Fonte: elaborado pelos autores, 2022.

Em outras palavras, o servidor do Redis está em estado ativo e é possível visualizar

algumas informações importantes, como a versão instalada na máquina, o modo de execução (*standalone*), a porta de conexão (*Port: 6379*) e o identificador único do processo (PID).

3.2 Exemplos Práticos

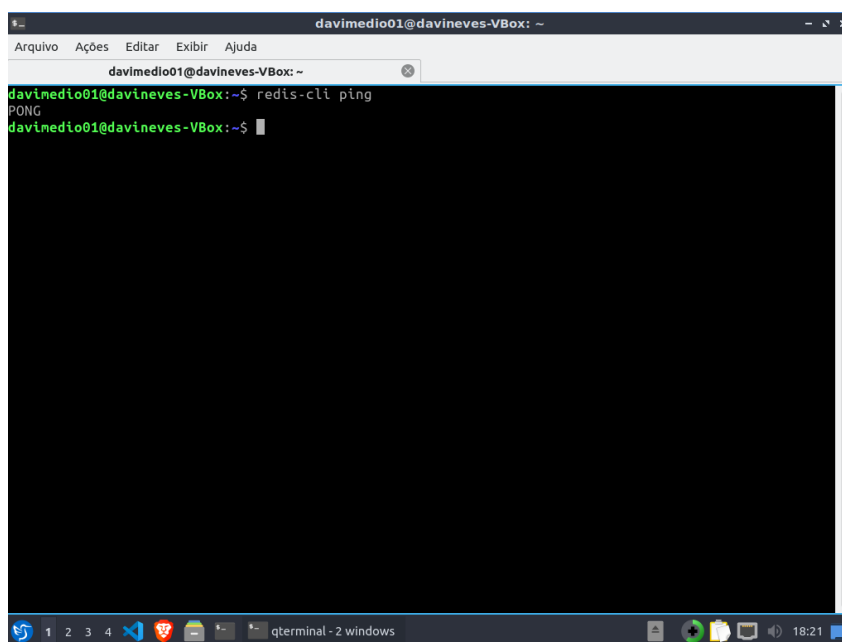
Para fins práticos, as configurações padrões de conexão do Redis serão utilizadas. Ou seja, os valores de "*localhost*" para endereço (ou "*127.0.0.1*", o qual se refere ao endereço local da máquina) e "*6379*" para porta de conexão. Considere, também, a condição de existência do último estado da seção anterior para os exemplos práticos, o qual representa a abertura e estado ativo do servidor do Redis na máquina.

3.2.1 Linha de Comando

Conforme descrito por Carlson (2013) e Labs (2022), existe uma ferramenta denominada "*redis-cli*" que, em outras palavras, simboliza um cliente interativo via terminal do sistema. Desta forma, é possível demonstrar as principais funcionalidades do Redis, como a criação e consulta de dados, em poucos minutos, por meio apenas da execução de um servidor Redis e da ferramenta citada.

Desta forma, o primeiro comando recomendado pela documentação consiste em executar o chamado "PING". Este comando realiza um teste de conexão com o servidor Redis ativo e, caso haja sucesso de conexão, é retornado a palavra "PONG" no próprio terminal, como mostra a imagem a seguir.

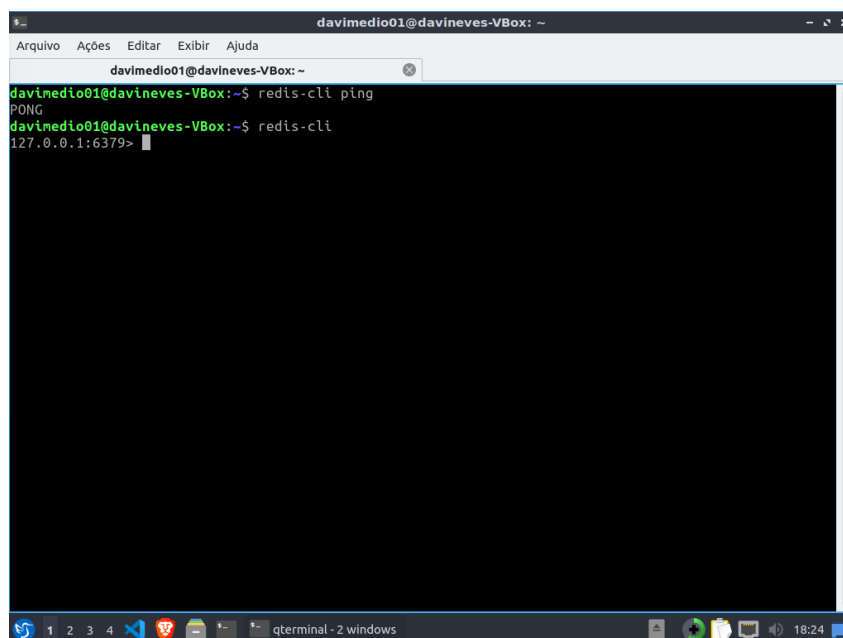
Figura 10 – Execução do *redis-cli*: teste de conexão por meio do comando PING.



Fonte: elaborado pelos autores, 2022.

Também, é possível abrir a ferramenta *redis-cli* para a execução contínua de comandos, como mostrado a seguir. Além disso, é possível visualizar o endereço de conexão e a porta em que serão realizados os comandos, como mostra em "127.0.0.1:6379".

Figura 11 – Execução do *redis-cli*: entrada para realização contínua de comandos.

A screenshot of a terminal window titled "davimedio01@davineves-VBox: ~". The terminal shows the following commands and output:

```
davimedio01@davineves-VBox:~$ redis-cli ping
PONG
davimedio01@davineves-VBox:~$ redis-cli
127.0.0.1:6379>
```

 The terminal window has a menu bar with "Arquivo", "Ações", "Editar", "Exibir", and "Ajuda". The bottom status bar shows "qterminal - 2 windows" and the time "18:24".

Fonte: elaborado pelos autores, 2022.

Como abordado em 2, o Redis possui cinco tipos de estruturas disponíveis para o armazenamento de dados. São elas: *string*, *list*, *set*, *hash*, *zset* (ou *sorted set*). Cada qual terá um exemplo adiante, abordando a inserção, a consulta e a remoção de dados.

- *String*: esse tipo consegue abordar valores do tipo texto, inteiros ou decimais.

Figura 12 – Execução do *redis-cli*: operações com o tipo *String*. Pode ser visto a inserção, consulta e remoção dos dados.

```

davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda

davimedio01@davineves-VBox: ~
davimedio01@davineves-VBox:~$ redis-cli ping
PONG
davimedio01@davineves-VBox:~$ redis-cli
127.0.0.1:6379> set chave_aqui valor_aqui
OK
127.0.0.1:6379> get chave_aqui
"valor_aqui"
127.0.0.1:6379> set 10 Nilceu
OK
127.0.0.1:6379> get 10
"Nilceu"
127.0.0.1:6379> del chave_aqui
(integer) 1
127.0.0.1:6379> del chave_aqui
(integer) 0
127.0.0.1:6379> get chave_aqui
(nil)
127.0.0.1:6379> del 10
(integer) 1
127.0.0.1:6379> get 10
(nil)
127.0.0.1:6379>

```

Fonte: elaborado pelos autores, 2022.

- *List*: esse tipo está relacionado com a utilização de uma lista ligada do tipo *String*.

Figura 13 – Execução do *redis-cli*: operações com o tipo *List*. Pode ser visto a inserção, consulta e remoção dos dados.

```

davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda

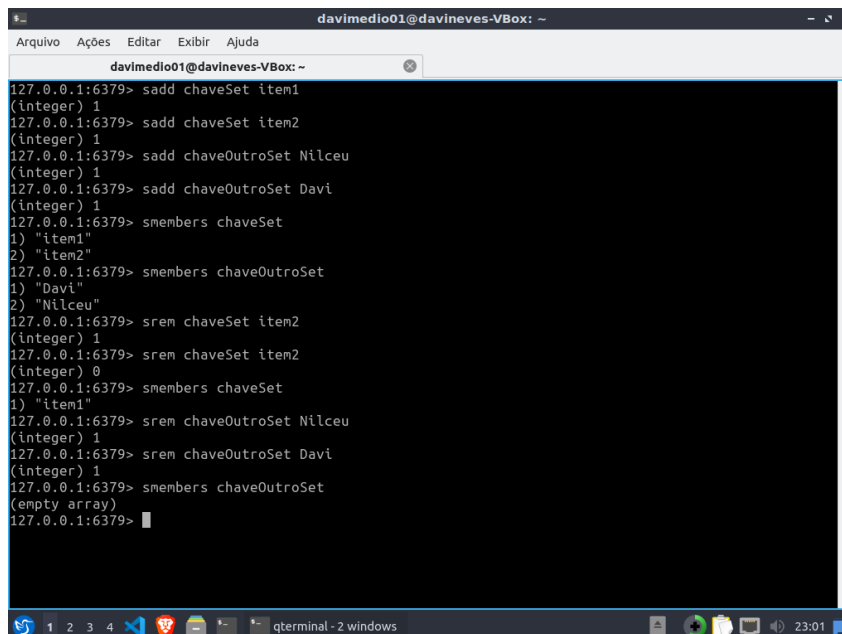
davimedio01@davineves-VBox: ~
127.0.0.1:6379> rpush list-key Davi
(integer) 1
127.0.0.1:6379> rpush list-key Davi2
(integer) 2
127.0.0.1:6379> rpush list-key Davi3
(integer) 3
127.0.0.1:6379> rpush list-key Nilceu
(integer) 4
127.0.0.1:6379> rpush list-key OutroNilceu
(integer) 5
127.0.0.1:6379> lrange list-key 0 -1
1) "Davi"
2) "Davi2"
3) "Davi3"
4) "Nilceu"
5) "OutroNilceu"
127.0.0.1:6379> rpush chave10 OutraLista
(integer) 1
127.0.0.1:6379> rpush chave10 OutroElemento
(integer) 2
127.0.0.1:6379> lrange chave10 0 1
1) "OutraLista"
2) "OutroElemento"
127.0.0.1:6379> lrange chave10 0 0
1) "OutraLista"
127.0.0.1:6379> lrange chave10 0 -1
1) "OutraLista"
2) "OutroElemento"
127.0.0.1:6379> lindex list-key 2
"Davi3"
127.0.0.1:6379> lindex list-key 0
"Davi"
127.0.0.1:6379> lindex chave10 1
"OutroElemento"
127.0.0.1:6379> lpop list-key
"Davi"
127.0.0.1:6379> lrange list-key 0 -1
1) "Davi2"
2) "Davi3"
3) "Nilceu"
4) "OutroNilceu"
127.0.0.1:6379>

```

Fonte: elaborado pelos autores, 2022.

- *Sets*: esse tipo consiste em uma coleção não ordenada de *Strings* únicas (sem repetição).

Figura 14 – Execução do *redis-cli*: operações com o tipo *Sets*. Pode ser visto a inserção, consulta e remoção dos dados.



```

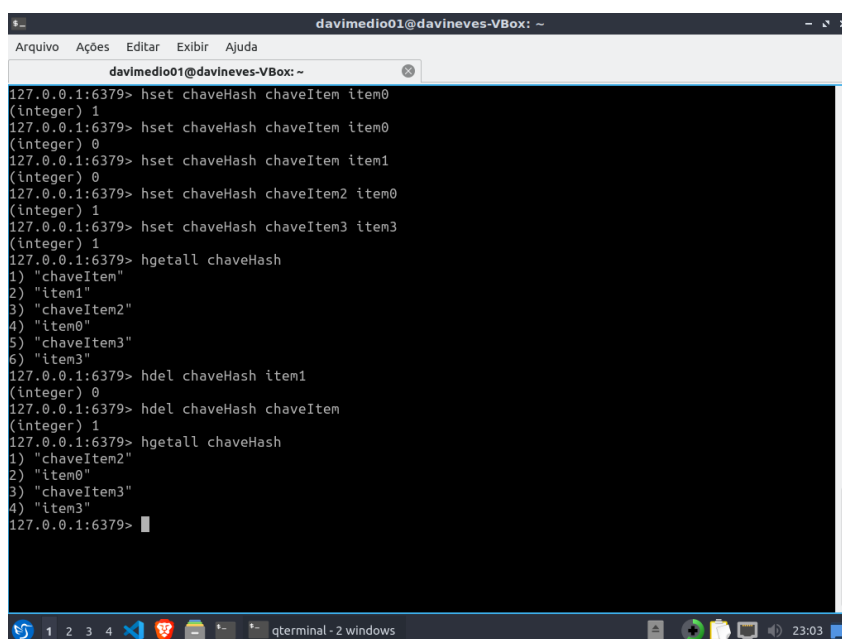
127.0.0.1:6379> sadd chaveSet item1
(integer) 1
127.0.0.1:6379> sadd chaveSet item2
(integer) 1
127.0.0.1:6379> sadd chaveOutroSet Nilceu
(integer) 1
127.0.0.1:6379> sadd chaveOutroSet Davi
(integer) 1
127.0.0.1:6379> smembers chaveSet
1) "item1"
2) "item2"
127.0.0.1:6379> smembers chaveOutroSet
1) "Davi"
2) "Nilceu"
127.0.0.1:6379> srem chaveSet item2
(integer) 1
127.0.0.1:6379> srem chaveSet item2
(integer) 0
127.0.0.1:6379> smembers chaveSet
1) "item1"
127.0.0.1:6379> srem chaveOutroSet Nilceu
(integer) 1
127.0.0.1:6379> srem chaveOutroSet Davi
(integer) 1
127.0.0.1:6379> smembers chaveOutroSet
(empty array)
127.0.0.1:6379>

```

Fonte: elaborado pelos autores, 2022.

- *Hashes*: utilização de uma tabela de *hashing* não ordenada que abrange os componentes chave-valor.

Figura 15 – Execução do *redis-cli*: operações com o tipo *Hashes*. Pode ser visto a inserção, consulta e remoção dos dados.



```

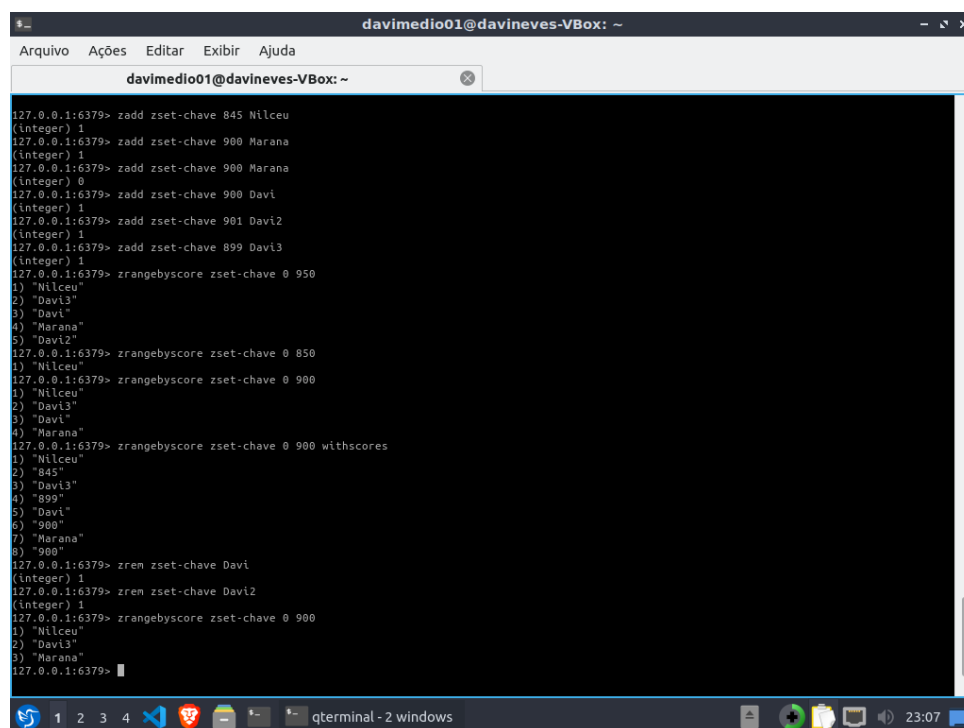
127.0.0.1:6379> hset chaveHash chaveItem item0
(integer) 1
127.0.0.1:6379> hset chaveHash chaveItem item0
(integer) 0
127.0.0.1:6379> hset chaveHash chaveItem item1
(integer) 0
127.0.0.1:6379> hset chaveHash chaveItem2 item0
(integer) 1
127.0.0.1:6379> hset chaveHash chaveItem3 item3
(integer) 1
127.0.0.1:6379> hgetall chaveHash
1) "chaveItem"
2) "item1"
3) "chaveItem2"
4) "item0"
5) "chaveItem3"
6) "item3"
127.0.0.1:6379> hdel chaveHash item1
(integer) 0
127.0.0.1:6379> hdel chaveHash chaveItem
(integer) 1
127.0.0.1:6379> hgetall chaveHash
1) "chaveItem2"
2) "item0"
3) "chaveItem3"
4) "item3"
127.0.0.1:6379>

```

Fonte: elaborado pelos autores, 2022.

- *Zset* ou *Sorted Sets*: consiste em um mapeamento ordenado de *Strings*, tendo essa ordenação dada por uma pontuação (chave).

Figura 16 – Execução do *redis-cli*: operações com o tipo *Zset*. Pode ser visto a inserção, consulta e remoção dos dados.



```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda

davimedio01@davineves-VBox: ~
127.0.0.1:6379> zadd zset-chave 845 Nilceu
(integer) 1
127.0.0.1:6379> zadd zset-chave 900 Marana
(integer) 1
127.0.0.1:6379> zadd zset-chave 900 Marana
(integer) 0
127.0.0.1:6379> zadd zset-chave 900 Davi
(integer) 1
127.0.0.1:6379> zadd zset-chave 901 Davi2
(integer) 1
127.0.0.1:6379> zadd zset-chave 899 Davi3
(integer) 1
127.0.0.1:6379> zrangebyscore zset-chave 0 950
1) "Nilceu"
2) "Davi3"
3) "Davi"
4) "Marana"
5) "Davi2"
127.0.0.1:6379> zrangebyscore zset-chave 0 850
1) "Nilceu"
127.0.0.1:6379> zrangebyscore zset-chave 0 900
1) "Nilceu"
2) "Davi3"
3) "Davi"
4) "Marana"
127.0.0.1:6379> zrangebyscore zset-chave 0 900 withscores
1) "Nilceu"
2) "845"
3) "Davi3"
4) "899"
5) "Davi"
6) "900"
7) "Marana"
8) "900"
127.0.0.1:6379> zrem zset-chave Davi
(integer) 1
127.0.0.1:6379> zrem zset-chave Davi2
(integer) 1
127.0.0.1:6379> zrangebyscore zset-chave 0 900
1) "Nilceu"
2) "Davi3"
3) "Marana"
127.0.0.1:6379>
```

Fonte: elaborado pelos autores, 2022.

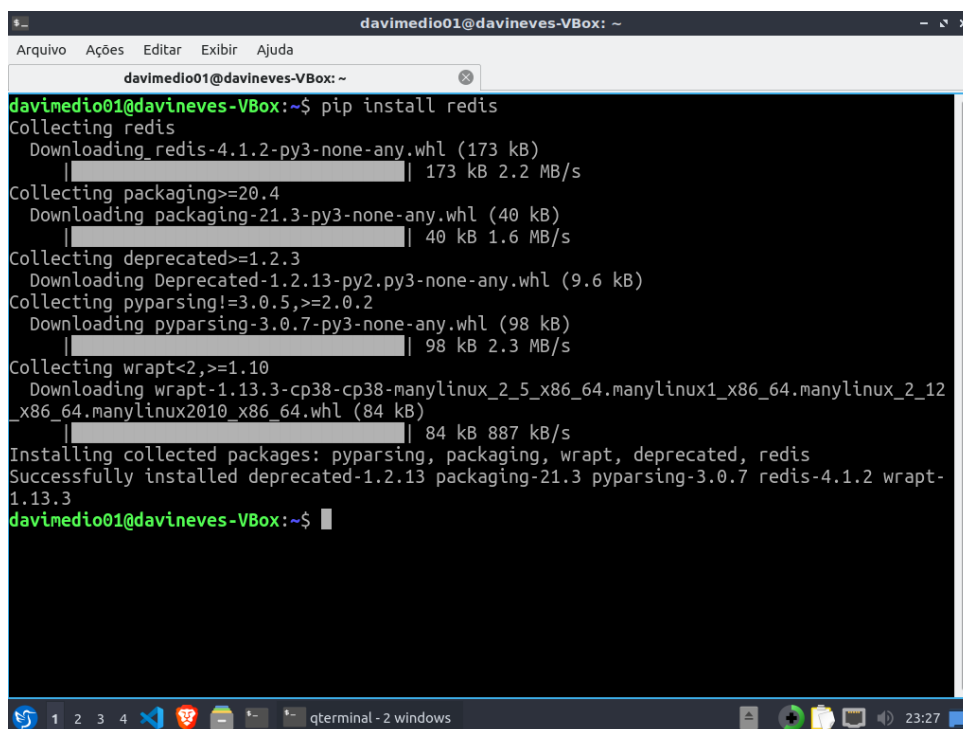
3.2.2 Aplicação em Python

Com relação as aplicações, é necessário um componente específico para a realização da conexão entre o cliente, ou seja, a aplicação, e o servidor do Redis. Esse componente é denominado conector e, de acordo com a Labs (2022), é disponibilizado diversos conectores oficiais e extraoficiais pela comunidade.

Especificamente, será construído uma aplicação em Python que utilizará o conector denominado *redis-py* (MACEDO; OLIVEIRA, 2011). Esse conector pode ser obtido oficialmente por meio do GitHub, disponível em: <https://github.com/redis/redis-py>. Portanto, além de ser um conector oficial desenvolvido pela Redis Labs, é totalmente de código-aberto, podendo ser mantido pela comunidade de desenvolvedores.

A instalação do conector *redis-py* é bem simples: basta executar o comando "pip install redis" no terminal do Ubuntu e aguardar o fim da instalação, como indica a imagem a seguir.

Figura 17 – Aplicação com Redis: instalação do conector para Python

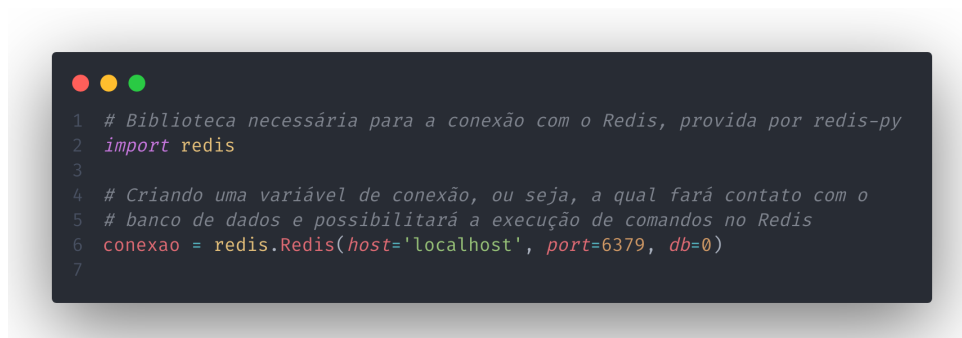


```
davimedio01@davineves-VBox: ~
Arquivo  Ações  Editar  Exibir  Ajuda
davimedio01@davineves-VBox: ~
davimedio01@davineves-VBox:~$ pip install redis
Collecting redis
  Downloading redis-4.1.2-py3-none-any.whl (173 kB)
    | 173 kB 2.2 MB/s
Collecting packaging>=20.4
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
    | 40 kB 1.6 MB/s
Collecting deprecated>=1.2.3
  Downloading Deprecated-1.2.13-py2.py3-none-any.whl (9.6 kB)
Collecting pyparsing!=3.0.5,>=2.0.2
  Downloading pyparsing-3.0.7-py3-none-any.whl (98 kB)
    | 98 kB 2.3 MB/s
Collecting wrapt<2,>=1.10
  Downloading wrapt-1.13.3-cp38-cp38-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux2010_x86_64.whl (84 kB)
    | 84 kB 887 kB/s
Installing collected packages: pyparsing, packaging, wrapt, deprecated, redis
Successfully installed deprecated-1.2.13 packaging-21.3 pyparsing-3.0.7 redis-4.1.2 wrapt-1.13.3
davimedio01@davineves-VBox:~$
```

Fonte: elaborado pelos autores, 2022.

A partir disso, é possível realizar o código-fonte da aplicação que utilizará o Redis como banco de dados. Para tanto, deve-se importar o pacote redis no início. Consequentemente, é necessário a criação de uma variável de conexão que, em outras palavras, tem a função de fazer o contato com o banco de dados e executar os comandos de adição, consulta e remoção de dados. A imagem a seguir demonstra esse processo.

Figura 18 – Aplicação com Redis: importação da biblioteca redis-py e criação da conexão.



```
1 # Biblioteca necessária para a conexão com o Redis, provida por redis-py
2 import redis
3
4 # Criando uma variável de conexão, ou seja, a qual fará contato com o
5 # banco de dados e possibilitará a execução de comandos no Redis
6 conexao = redis.Redis(host='localhost', port=6379, db=0)
7
```

Fonte: elaborado pelos autores, 2022.

Após isso, é possível utilizar funções pré-definidas para manipular os diferentes tipos de objetos do Redis pelo Python. A primeira imagem a seguir traz um exemplo de cada tipo de dados, ao passo que a segunda imagem apresenta os resultados da execução do código.

Figura 19 – Aplicação com Redis: manipulação no Python para cada tipo de dado do Redis.

```

8 # A partir daqui, é possível executar comandos de adição, consulta e remoção,
9 # assim como mostrados na seção de "Linha de Comandos"
10
11 # Exemplo com a estrutura STRING
12 conexao.set('chave', 'valor') # Inserir Dado
13 print(conexao.get('chave')) # Consultar Dado
14 conexao.delete('chave') # Remover Dado
15 print(conexao.get('chave')) # Consultar Dado Removido
16
17 # Exemplo com a estrutura LIST
18 conexao.lpush('nomeLista', 'item') # Inserir Um Dado no início da lista
19 conexao.rpush('nomeLista', 'item2') # Inserir Um Dado no fim da lista
20 conexao.lpush('nomeLista', 'item3', 'item4') # Inserir Vários Dados no início
21 # Consultar Lista por meio de LPOP (Remoção do início da lista)
22 while(conexao.llen('nomeLista') != 0):
23     print(conexao.lpop('nomeLista'))
24
25 # Exemplo com a estrutura SET
26 conexao.sadd('nomeSet', 'valor') # Inserir Dado
27 conexao.sadd('nomeSet', 'valor2') # Inserir Dado
28 print(conexao.smembers('nomeSet')) # Consultar todos elementos do Set
29 conexao.srem('nomeSet', 'valor2') # Remover Dado
30 print(conexao.smembers('nomeSet')) # Consultar todos elementos do Set
31
32 # Exemplo com a estrutura HASH
33 conexao.hset('nomeTabelaHash', 'chave1', 'item1') # Inserir Dado
34 conexao.hset('nomeTabelaHash', 'chave2', 'item2') # Inserir Dado
35 print(conexao.hvals('nomeTabelaHash')) # Consultar Dados
36 print(conexao.hkeys('nomeTabelaHash')) # Consultar Chaves
37 print(conexao.hgetall('nomeTabelaHash')) # Consultar Dados e Chaves
38 conexao.hdel('nomeTabelaHash', 'chave1') # Remover Dado
39 print(conexao.hgetall('nomeTabelaHash')) # Consultar Dados e Chaves
40
41 # Exemplo com a estrutura SORTED SET
42 conexao.zadd('nomeSortedSet', {'item1': 850}) # Inserir Dado com Score = 850
43 conexao.zadd('nomeSortedSet', {'item2': 899}) # Inserir Dado com Score = 899
44 conexao.zadd('nomeSortedSet', {'item3': 901}) # Inserir Dado com Score = 901
45 conexao.zadd('nomeSortedSet', {'item4': 900}) # Inserir Dado com Score = 900
46 print(conexao.zrange('nomeSortedSet', 0, -1)) # Consultar Todos os Dados
47 print(conexao.zrangebyscore('nomeSortedSet', min=0, max=899)) # Consultar: 0 < Score < 900
48 conexao.zrem('nomeSortedSet', 'item1', 'item3') # Remover Dados
49 print(conexao.zrange('nomeSortedSet', 0, -1)) # Consultar Todos os Dados
50

```

Fonte: elaborado pelos autores, 2022.

Figura 20 – Aplicação com Redis: resultado de execução do código.

```

davinedio01@davineves-VBox: ~/Downloads
davinedio01@davineves-VBox: ~/Downloads$ python3 AplicaçãoConRedis_Exemplo.py
b'valor'
None
b'item4'
b'item3'
b'item'
b'item2'

{b'valor2', b'valor'}
{b'valor'}

[b'item2', b'item1']
[b'chave2', b'chave1']
{b'chave2': b'item2', b'chave1': b'item1'}
{b'chave2': b'item2'}

[b'item1', b'item2', b'item4', b'item3']
[b'item1', b'item2']
[b'item2', b'item4']
davinedio01@davineves-VBox: ~/Downloads$

```

Fonte: elaborado pelos autores, 2022.

4 Metodologia

O trabalho em questão apresenta uma revisão bibliográfica a respeito do armazenamento de estrutura de dados de chave-valor Redis, com enfoque especial em sua criação e aplicações. Desta forma, foram abordados diferentes tópicos relacionados a bancos de dados, constituindo: o que são bancos de dados; abordagens de bancos de dados relacionais e não relacionais, o porquê de suas origens e comparação entre ambas, com foco nas vantagens e desvantagens de cada uma; exposição dos tipos de bancos de dados NoSQL; introdução ao banco de dados Redis e uma breve revisão de sua história e surgimento, englobando sua implementação em situações e sistemas no mundo real a partir de exemplos de casos de uso; um capítulo especial, intitulado "tutorial", visando guiar o leitor na instalação do suporte para Redis em seu sistema, inicialização do servidor Redis, introdução aos comandos mais básicos da linguagem e construção de um programa simples para criação do banco de dados e inserção e manipulação de elementos nele.

Para tanto, a monografia está baseada em caráter de pesquisa bibliográfica que, segundo Pizzani et al. (2012), consiste na realização de um levantamento de informações sobre o tema com base em diversas fontes, como livros, periódicos, artigos de jornais, sites da Internet, dentre outros. As autoras ainda expressam que esse tipo de pesquisa possui diversos objetivos, tais quais proporcionar um aprendizado acerca de determinada área do conhecimento (PIZZANI et al., 2012). Por consequência, o presente documento serve como uma fonte de referência para que os interessados em aprofundar seus conhecimentos em banco de dados e aprender novas tecnologias possam entender como funciona o Redis.

Com base no exposto, evidencia-se abaixo o processo de coleta e análise de dados que proporcionou a explicação direta e de fácil entendimento sobre as componentes citadas.

4.1 Coleta e Análise de Dados

O processo para coleta de dados ocorreu em três etapas. A primeira etapa consistiu em uma pesquisa por publicações científicas a respeito de bancos de dados relacionais e não relacionais, como são arquitetados e seu funcionamento, as diferenças entre cada tratamento, as vantagens e desvantagens em empregar um ou outro em um sistema e exemplos de suas aplicações através da ferramenta de busca Google Scholar, nos retornando resultados nas seguintes bases de dados: IEEE Xplore, CiteSeerX, Research Gate e Asian Journal of Research in Computer Science. Aprofundando a pesquisa nas bases citadas, com enfoque nas bases IEEE Xplore e Research Gate, obteve-se como resultado um alto número de artigos condizentes com os parâmetros do estudo, corroborando com nossa justificativa no capítulo 1. Segue abaixo uma tabela com os resultados encontrados em cada base, considerando as palavras-chave "SQL",

"NoSQL", "Database" e "Redis" e favorecendo artigos publicados nos últimos três anos.

Quadro 1 – Comparativo de resultados em quatro diferentes bases de dados

Base de Dados	Palavras-chave	Período de Publicação	Número de Resultados
Google Scholar	SQL; NoSQL; Database; Redis	Últimos três anos: entre 2019 e 2022	1.670
IEEE Xplore Digital Library	SQL; NoSQL; Database; Redis	Últimos três anos: entre 2019 e 2022	200
Research Gate	SQL; NoSQL; Database; Redis	Últimos três anos: entre 2019 e 2022	100

Fonte: Elaborado pelos autores, 2022.

Diante os achados, por ordem de relevância e adequação com o proposto em nosso estudo, efetuou-se uma análise e filtração minuciosa dos resultados obtidos. Observou-se que não houveram estudos bibliográficos semelhantes ao presente trabalho, ou seja, apresentando e discorrendo acerca da tecnologia. No entanto, houveram artigos com conceitos e fundamentos essenciais para o entendimento da operação do Redis. Nessa fase, quatro artigos foram selecionados para compor grande parte do trabalho, sendo eles:

- ALI, W.; SHAFIQUE, M. U.; MAJEED, M. A.; RAZA, A. Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics. Asian Journal of Research in Computer Science, p. 1–10, out. 2019. ISSN 2581-8260.
- KHASAWNEH, T. N.; AL-SAHLEE, M. H.; SAFIA, A. A. Sql, newsql, and nosql databases: A comparative survey. In: 2020 11th International Conference on Information and Communication Systems (ICICS). [S.l.: s.n.], 2020. p. 013–021.
- NAYAK, A.; PORIYA, A.; POOJARY, D. Article: Type of nosql databases and its comparison with relational databases. International Journal of Applied Information Systems, v. 5, n. 4, p. 16–19, March 2013. Published by Foundation of Computer Science, New York, USA.
- SAREEN, P.; KUMAR, P. Nosql database and its comparison with sql database. International Journal of Computer Science & Communication Networks, v. 5, n. 5, p. 293–298, 2015.

A segunda fase da pesquisa referiu-se a busca pela documentação oficial do Redis, encontrada no portal Redis (LABS, 2022), hospedado no site <https://redis.io/>. O endereço centraliza as novidades de cada versão lançada através de um *feed* denominado Redis News. Para mais, a documentação completa do Redis é fornecida, incluindo uma lista de comandos implementados por ele, que constituem sua linguagem de consulta semelhante a SQL, apresentação dos módulos e API disponibilizados para execução em projetos particulares, tutoriais e respostas a dúvidas frequentes, ensinando como começar a usar as funcionalidades da tecnologia e sanando as questões mais complexas em lidar com ele, quesitos como administração, *benchmarks* de performance e uma página específica para resolução de problemas. Links para o download dos lançamentos de versão estável, mais recente e versões antigas, canais de comunicação da comunidade e suporte também são disponibilizados.

Na terceira e última etapa, a partir da recomendação de literatura sobre Redis providenciada pela documentação, procurou-se por livros que delineassem seu funcionamento em detalhes, intrinsecamente, possuindo a função de um guia prático, explicando o básico da tecnologia, desde seus tipos de dados suportados, como realizar comandos e manipulá-los, melhores práticas e afins. Ademais, a base de dados SpringerLink também foi utilizada para encontrar livros gratuitos sobre bancos de dados SQL e NoSQL e arquitetura da ferramenta. Alguns livros foram selecionados e, no caso de livros pagos, realizou-se a obtenção destes.

Após a coleta dos dados, os materiais foram analisados, com o propósito de averiguar o casamento com as metas do trabalho em questão. De todos os materiais levantados, somente uma pequena parcela serviu como base teórica. Tendo como base esta seleta parcela, efetivou-se a criação de diferentes tópicos a respeito do tema: banco de dados relacionais e não relacionais, o porquê de banco de dados não relacionais terem emergido, comparação entre SQL e NoSQL e Banco de Dados Redis, incorporando sua história e operação.

5 Considerações Finais

O objetivo deste trabalho foi reunir e apresentar as informações mais relevantes a respeito do Redis, um famoso banco de dados não relacional. Ademais, foi necessário a apresentação de alguns exemplos práticos, tendo em vista facilitar o entendimento do leitor.

Além disso, destaca-se a importância e relevância deste tema perante a comunidade *open-source*, uma vez que o Redis tem crescido exponencialmente desde o seu surgimento e, assim, torna-se necessário a criação de um material que trouxesse as informações necessárias para o início do desenvolvimento com esse banco de dados. Desta forma, é possibilitado uma maior difusão das informações corretas e mínimas a respeito dessa importante ferramenta.

Constata-se, inicialmente, que o primeiro passo do trabalho consistiu em encontrar fontes de pesquisa que garantissem a confiabilidade e relevância perante o tema. Ademais, destacou-se a existência de quatro artigos principais que trouxeram diversas informações de comparação e explicação a respeito dos bancos de dados não relacionais, contexto em que o Redis se insere. Ainda, as informações foram reforçadas pela própria documentação, a qual possui uma indicação de livros extraoficiais acerca do tema e, desta forma, foram utilizados para compor a base da pesquisa. Motores de pesquisas acadêmicos, como o Google Scholar e o Research Gate, também foram devidamente utilizados para encontrar outros artigos mais antigos, mas que fossem extremamente relevantes e válidos para o tema atualmente.

A partir disso, os dados pesquisados tiveram todas suas informações filtradas e concatenadas, de tal forma que as ideias fossem apresentadas de maneira direta e de fácil entendimento ao leitor. Para tanto, o presente trabalho se dividiu na seguinte maneira: introdução, contendo informações iniciais a respeito das características, vantagens e desvantagens entre banco de dados relacional e não relacional, como o Redis; revisão de literatura, a qual contém toda a parte teórica do trabalho, detalhando as informações necessárias para a compreensão das características principais do Redis; tutorial, sendo um capítulo exclusivo para instruir o leitor em como instalar e utilizar o Redis em sua máquina; e, por fim, a metodologia, a qual consistiu em explicar quais foram as principais referências e materiais utilizados na composição do trabalho.

Diante de tudo isso, é possível afirmar que os objetivos foram devidamente cumpridos, uma vez que foram apresentadas, de maneira direta, todas as informações necessárias para o entendimento do banco de dados não relacional Redis. Ademais, pode-se afirmar que, com o presente trabalho, será possível realizar uma maior e correta difusão das informações acerca do tema e, com isso, espera-se o surgimento de mais desenvolvedores nesta importante área.

Referências

- ALI, W.; SHAFIQUE, M. U.; MAJEED, M. A.; RAZA, A. Comparison between SQL and NoSQL Databases and Their Relationship with Big Data Analytics. *Asian Journal of Research in Computer Science*, p. 1–10, out. 2019. ISSN 2581-8260. Disponível em: <http://www.journalajrcos.com/index.php/AJRCOS/article/view/30108>. Acesso em: 04 fev. 2022.
- CARLSON, J. *Redis in Action*. [S.l.]: Simon and Schuster, 2013.
- ELMASRI, R.; NAVATHE, S. *Sistemas de Banco de Dados*. [S.l.: s.n.], 2011. 808 p.
- HEUSER, C. A. *Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS*. [S.l.]: Bookman Editora, 2009.
- KHASAWNEH, T. N.; AL-SAHLEE, M. H.; SAFIA, A. A. Sql, newsql, and nosql databases: A comparative survey. In: *2020 11th International Conference on Information and Communication Systems (ICICS)*. [S.l.: s.n.], 2020. p. 013–021.
- LABS, R. *Redis Documentation*. 2022. Disponível em: <https://redis.io/documentation>. Acesso em: 04 fev. 2022.
- MACEDO, T.; OLIVEIRA, F. *Redis Cookbook: Practical techniques for fast data manipulation*. [S.l.]: "O'Reilly Media, Inc.", 2011.
- MEIER, A.; KAUFMANN, M. *SQL & NoSQL databases*. [S.l.]: Springer, 2019.
- NAYAK, A.; PORIYA, A.; POOJARY, D. Article: Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, v. 5, n. 4, p. 16–19, March 2013. Published by Foundation of Computer Science, New York, USA.
- PIZZANI, L.; SILVA, R. C. da; BELLO, S. F.; HAYASHI, M. C. P. I. A arte da pesquisa bibliográfica na busca do conhecimento. *RDBCI: Revista Digital de Biblioteconomia e Ciência da Informação*, v. 10, n. 2, p. 53–66, 2012.
- RUSSO, M. *Redis, from the Ground Up*. 2010. Disponível em: <https://mjrusso.com/redis-from-the-ground-up/>. Acesso em: 04 fev. 2022.
- SADALAGE, P. J.; FOWLER, M. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. [S.l.]: Pearson Education, 2013.
- SANFILIPPO, S. <antirez> blog. [entre 2012 e 2021]. Disponível em: <http://antirez.com/>. Acesso em: 04 fev. 2022.
- SAREEN, P.; KUMAR, P. Nosql database and its comparison with sql database. *International Journal of Computer Science & Communication Networks*, v. 5, n. 5, p. 293–298, 2015.
- SERVICES, A. W. *Redis: armazenamento de dados na memória. Como funciona e porque você deve usá-lo*. 2022. Disponível em: <https://aws.amazon.com/pt/redis/>. Acesso em: 04 fev. 2022.
- SILVA, M. D.; TAVARES, H. *Redis Essentials*. [S.l.]: Packt Publishing Ltd, 2015.