Pontifícia Universidade Católica de Goiás

Fórum Avaliativo
Banco de Dados para Big Data

Curso: Big Data e Inteligência Artificial

João Paulo Moreira Rosa

Igor Guimarães de Oliveira

Thiago Alves da Silva

Luiz Paulo T. Gonçalves

Goiânia – 2023

1. Qual o tipo do banco de dados ?

Redis é um acrônimo de REmote DIctionary Server (servidor de dicionário remoto). Por conta da sua velocidade e facilidade de uso, o Redis é uma escolha em alta demanda para aplicações web e móveis, como também de jogos, tecnologia de anúncios e IoT, que exigem o melhor desempenho do mercado.

2. Qual estratégia de armazenamento ele implementa ?

O Redis implementa uma estratégia de armazenamento chave-valor, onde cada item armazenado no banco de dados é identificado por uma chave exclusiva. Os valores armazenados no Redis podem ser de diferentes tipos de dados, incluindo strings, hashes, listas, conjuntos e sorted sets.

Para simplificar, podemos pensar em algo como várias gavetas, onde cada gaveta possui um nome, que não pode se repetir e dentro de cada gaveta possui um valor que representa a mesma. Por exemplo, cada gaveta representa o nome de uma pessoa e dentro dela possui o valor da idade, então se eu quero saber a idade do Lucas, vou até a gaveta “Lucas” e pego o valor, no caso, 22. Simples, não é?

Este exemplo foi exatamente a estrutura mais simples de armazenamento do Redis, onde temos uma chave e um valor, podendo ser uma letra, palavra, texto ou número. Só que o Redis vai além disso, pois o valor da chave não precisa ser necessariamente algo tão simples como o exemplo anterior, o valor pode ser uma estrutura de dados um pouco mais complexa, como uma lista, conjunto ou hash.

3. Em qual das combinações do dentro do teorema CAP ele melhor se enquadra ?

De acordo com o teorema CAP (Consistency, Availability, Partition Tolerance), um sistema distribuído de banco de dados só pode garantir duas das três propriedades: consistência, disponibilidade e tolerância a partições.

O Redis é conhecido por sua alta disponibilidade e tolerância a partições, mas não fornece garantias fortes de consistência em todos os casos. Portanto, o Redis geralmente é classificado como um banco de dados "AP", o que significa que prioriza disponibilidade e tolerância a partições em vez de consistência. No entanto, o Redis tem recursos para garantir a consistência em casos específicos, como a utilização do mecanismo de transações e outras técnicas avançadas de manipulação de dados. Em resumo, o Redis é altamente disponível e tolerante a falhas, mas a consistência precisa ser gerenciada cuidadosamente dependendo dos requisitos da aplicação.

4. Para qual tipo de aplicações ele é mais indicado ?

Um outro caso que é possível utilizar o Redis, é quando a página inicial de um site exibe totalizadores ou um ranking de games que muitas vezes não são dados que atualizam a todo momento e muitas vezes ao tentar recuperar estas informações diretamente de uma base relacional e realizar uma soma, média ou qualquer que seja a operação, demanda uma certa quantidade de recursos, prejudicando ainda mais a performance caso tenha um alto volume de acesso a estas informações.

Sendo assim, para economizar recursos e consequentemente retornar estas informações com maior velocidade, é possível armazenar estes totalizadores diretamente no Redis, assim, quando o usuário acessar a página, não terá que recuperar uma quantidade de registros, depois fazer uma operação e retornar, mas já poderá recuperar estes valores diretamente.

Demais aplicações:

1. Armazenamento em cache de dados: O Redis é frequentemente usado como um cache em aplicativos web para acelerar a recuperação de dados.
2. Filas de mensagens: O Redis pode ser usado como uma fila de mensagens para enviar mensagens entre processos, ou entre sistemas distribuídos.
3. Sessões de usuário: O Redis pode ser usado para armazenar e gerenciar informações de sessão de usuário em aplicativos da web.
4. Análise de dados em tempo real: O Redis pode ser usado para armazenar e processar dados em tempo real, como contagem de visualizações de página, atualizações de estoque, etc.
5. Armazenamento de metadados: O Redis pode ser usado para armazenar informações de metadados, como informações de configuração do sistema, chaves de API, etc.
6. Aplicativos de jogos: O Redis é frequentemente usado em aplicativos de jogos para armazenar dados de jogo em tempo real, como pontuações, status de jogadores, etc.
7. Sistemas de votação: O Redis é frequentemente usado em sistemas de votação para gerenciar o processo de votação em tempo real.
8. Sistemas de análise de dados: O Redis pode ser usado como uma camada intermediária para gerenciar e processar grandes volumes de dados antes de serem armazenados em um banco de dados permanente.

Segue o anexo do relatório:

Anexo

Um tutorial explicando como baixar, instalar e fazer as configurações iniciais para uso no ambiente Linux, Ubuntu 20.04

Instalação do Redis

```
# sudo apt update
# sudo apt upgrade
# sudo apt install redis-server # instalar
```

Depois de concluir a instalação, o Redis será iniciado automaticamente. Para verificar se o Redis está sendo executado, você pode executar o seguinte comando:

```
# sudo systemctl status redis-server
```

Configurando o Redis no ambiente Linux/Ubuntu 20.04

O Redis usa a porta padrão 6379 para comunicação, verifique se a porta está aberta executando o seguinte comando:

```
# sudo ufw allow 6379/tcp
```

Isso permite que o tráfego da porta 6379 passe pelo firewall do Ubuntu. O comando sudo ufw allow 6379/tcp permite que o tráfego na porta TCP 6379 seja permitido através do firewall UFW (Uncomplicated Firewall). Quando você executou o comando, foi solicitada a sua senha de administrador do sistema e, em seguida, o UFW adicionou uma regra para permitir o tráfego na porta 6379.

O output que você recebeu informa que as regras do firewall foram atualizadas para as conexões IPv4 e IPv6, que são as versões do protocolo de internet. Isso significa que tanto as conexões IPv4 quanto as IPv6 agora podem passar pelo firewall na porta TCP 6379

```
# sudo snap install redis-desktop-manager
```

Utilizando Redis via linguagem R

Pacotes/dependências utilizadas

```
rm(list = ls())

# Packages/Dependências

library(tidyverse)
library(redux)
library(CliometricsBR)
```

Importando dados para envio

```
# Dados para enviar para o banco de dados
# Dados via pacote CliometricsBR - dados de exportação de açúcar e café

export_raw <- CliometricsBR::get_seriesIPEA(codes = c("HIST_XACUCARQ",
                                                      "HIST_XCAFEQ"))

#metadados <- CliometricsBR::get_metadadosIPEA(codes = "all")
# Limpando e organizando o data.frame
```

Limpando e organizando dataset

```
db_export <- export_raw %>%
             dplyr::relocate(date, .after = NULL) %>%
             pivot_wider(id_cols = date,
                         names_from = code_series,
                         values_from = value) %>%
             janitor::clean_names() %>%
             dplyr::rename("Açúcar" = "hist_xacucarq",
                           "Café" = "hist_xcafeq")
```

Conectado ao Redis

```
# Conectando ao Redis - Cliometrics_Prod --------------------------------------

redis_conn <- redux::redis_connection(redis_config())
print(redis_conn)
```

```
## <redis_connection[redux]>:
##   - config()
##   - reconnect()
##   - command(cmd)
##   - pipeline(cmds)
##   - subscribe(channel, pattern, callback, envir = parent.frame())
```

```
r_conex <- redux::hiredis()
r_conex$PING()
```

```
## [Redis: PONG]
```

OPERAÇÃO DE INSERÇÃO NO BANCO DE DADOS REDIS

```
coffe_raw_serialized <- base::serialize(db_export %>%
                                          select(-"Açúcar"), NULL)

r_conex$SET("export_cafe", coffe_raw_serialized) # Enviar os dados de café
```

```
## [Redis: OK]
```

```
sugar_raw_serialized <- base::serialize(db_export %>%
                                           select(-"Café"), NULL)

r_conex$SET("export_acucar", sugar_raw_serialized)  # Enviar os dados de Açúcar
```

```
## [Redis: OK]
```

OPERAÇÃO DE RECUPERAÇÃO/SELEÇÃO

```
check_redis <- function(db = as.character()){

  db_export_retrieved <- base::unserialize(r_conex$GET(db))
  db_export_retrieved %>% head()

}
```

Verificar se os dados foram recuperados corretamente

```
check_redis("export_acucar")
```

```
## # A tibble: 6 x 2
##   date       Açúcar
##   <date>      <dbl>
## 1 1821-01-01  35168
## 2 1822-01-01  36694
## 3 1823-01-01  53549
## 4 1824-01-01  44976
## 5 1825-01-01  35485
## 6 1826-01-01  35410
```

```
check_redis("export_cafe")
```

```
## # A tibble: 6 x 2
##   date        Café
##   <date>     <dbl>
## 1 1821-01-01   129
## 2 1822-01-01   186
## 3 1823-01-01   226
## 4 1824-01-01   274
## 5 1825-01-01   224
## 6 1826-01-01   318
```

OPERAÇÃO DE REMOÇÃO

```
r_conex$DEL("export_acucar")
```

```
## [1] 1
```

```
r_conex$DEL("export_cafe")
```

```
## [1] 1
```