Trabalho Prático: Banco de Dados Chave-Valor *

Prof. Pedro Henrique Penna

Graduação em Engenharia de Software
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

1. Contexto

Um banco de dados de chave-valor usa um dicionário para armazenar dados, como ilustrado na Figura 1. Nessa organização, dados são estruturados em registros contendo dois campos: uma chave que identifica unicamente aquele registro no banco de dados; e um valor que armazena o dado propriamente dito. Chaves e valores podem ter tipos quaisquer e registros não possuem nenhuma relação entre si. Portanto, programas possuem máxima flexibilidade para gerenciar dados armazenados. Tipicamente, as seguintes operações são fornecidas:

- Inserção: insere um novo objeto no banco de dados.
- Remoção: remove um objeto do banco de dados.
- Busca: recupera um objeto no banco de dados.
- Atualização: atualiza um determinado objeto no banco de dados.

Bancos de dados de chave-valor são altamente particionáveis e permitem maior escalabilidade horizontal, em contrapartida a banco de dados relacionais. Por esse motivo, banco de dados de chave-valor têm ganhado cada vez mais espaço em aplicações atuais, principalmente no contexto da *cloud*. Alguns exemplos e banco dedados de chave-valor amplamente utilizados são o Redis, DynamoDB, LevelDB and MemcachedDB.

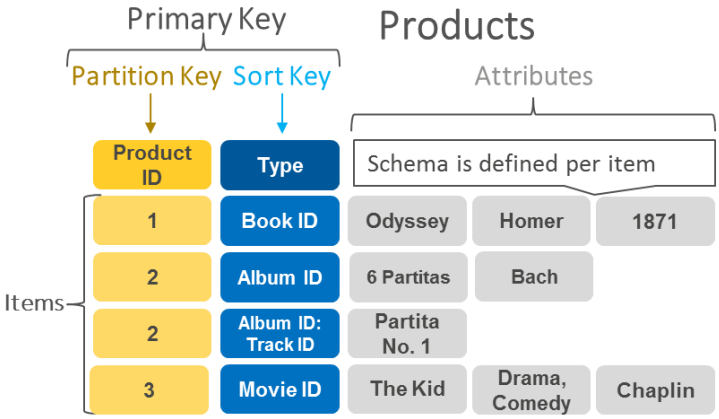

Figura 1: Arquitetura do DynomoDB.

* O presente enunciado está sujeito a correções e alterações.

2. Descrição

Nesse trabalho você deverá desenvolver um banco de dados de chave-valor com os seguintes requisitos funcionais:

- Inserir / Remover / Buscar / Atualizar objetos no banco de dados. Objetos podem ser de qualquer tipo (primitivo ou composto). Porém, o identificador de um registro deve ser obrigatoriamente do tipo inteiro sem sinal.
- Listar / Filtrar objetos no banco de dados. Objetos no banco de dados podem ser listados e/ou filtrados segundo uma chave de ordenação e um critério informada pela aplicação cliente. Essa chave deve ser necessariamente do tipo inteiro sem sinal.
- Persistir em um arquivo um histórico de operações no banco de dados.

Além dos requisitos funcionais listados anteriormente, o seu banco de dados de chave-valor deve apresentar os seguintes requisitos não funcionais:

- Adotar uma arquitetura multithread. Nessa arquitetura, ao menos duas threads devem ser criadas para atender às requisições no banco de dados (*ie.* inserção, remoção, buscar, atualização, listagem e filtragem) e uma outra thread para escrever no arquivo de histórico as operações realizadas no banco de dados.
- Possibilitar a inter-operação com demais processos usando algum dos seguintes mecanismos de comunicação: *named pipes*, *named message queues* ou *sockets*.
- Escrever um programa de testes que demonstra a inter-operação com o banco de dados desenvolvido.

3. Especificações Técnicas

Você é livre para definir a interface de inter-operação do seu banco de dados com os demais processos. No entanto, o banco de dados deve suportar a seguinte interface de linha de comando para possibilitar uma depuração de suas funcionalidades essenciais:

`simpledb [cmd]`
Manipula registros do banco de dados de chave-valor SimpleDB usando a operação especificada por `cmd`. As seguintes operações são suportadas:

`--insert=<sort-key,value>`
Insere um objeto no banco de dados. A chave de ordenação (`sort-key`) é um inteiro positivo. O objeto é codificado em uma cadeia de caracteres pela aplicação cliente. Ao concluir a operação, a chave do objeto inserido é impresso na saída padrão. Operações realizadas são escritas no arquivo `db.log`.

`--search=<key>`
Busca no banco de dados objeto o que é identificado pela chave `key`. Caso o objeto seja encontrado, o objeto (codificado por uma cadeia de caracteres) e sua chave de ordenação são impressos na saída padrão. Operações realizadas são escritas no arquivo `db.log`.

`--remove=<key>`
Remove do banco de dados o objeto que é identificado pela chave `key`. Operações realizadas são escritas no arquivo `db.log`.

`--update=<key,sort-key,value>`
Atualiza o objeto que é identificado pela chave `key`. A chave de ordenação (`sort-key`) é um inteiro positivo. O objeto é codificado em uma cadeia de caracteres pela aplicação cliente. Operações realizadas são escritas no arquivo `db.log`.

`--list=<expr> | --reverse-list=<expr>`
Lista em ordem crescente (`--list`) ou em ordem decrescente (`--reverse-list`) todos os objetos que possam uma chave de ordenação que atenda aos critérios especificados pela expressão `expr`. Operações realizadas são escritas no arquivo `db.log`. A expressão `expr` deve aceitar qualquer operação lógica simples envolvendo a chave:

- `key>X`: objetos que possuem chave de ordenação maior que `X`.
- `key<X`: objetos que possuem chave de ordenação menor que `X`.
- `key=X`: objetos que possuem chave ordenação igual a `X`.
- `key>=X`: objetos que possuem chave de ordenação maior ou igual que `X`.
- `key<=X`: objetos que possuem chave de ordenação menor ou igual que `X`.

4. Entrega

O programa a ser desenvolvido em uma das seguintes linguagens: `C`, `C++`, `GoLang` ou `Rust`. Você deve entregar o código fonte e outros artefatos produzidos.

O projeto deverá ser necessariamente desenvolvido usando o sistema de versionamento Git. Para hospedar a árvore de fontes, qualquer plataforma de acesso público, como o GitHub, BitBucket ou então GitLab, pode ser usada.

Na árvore de fontes do projeto, informações suficientes para a compilação do programa devem ser fornecidas. Obrigatoriamente, a compilação deve suportar o ambiente Linux Ubuntu 20.04 e não deve exigir a instalação de pacotes e/ou programas de terceiros (*ie.* IDEs). Portanto, recomenda-se que seja usado um sistema de compilação independente de plataforma, como o `make` ou `cmake` (veja a Seção de Distribuição de Pontos Extras).

5. Distribuição de Pontos

Este trabalho deve ser desenvolvido em grupo de três a quatro integrantes. O link do repositório Git contendo a árvore de fontes do projeto deverá ser entregue em um arquivo texto, que deve ser depositado em uma pasta no SGA antes do prazo para entrega estipulado. *Commits* realizados no repositório após o prazo de entregue ido trabalho serão desconsiderados. Esse trabalho será avaliado da seguinte forma:

- Corretude da Solução (60% dos pontos)
- Implementação de Testes (20% dos pontos)

- Qualidade e Documentação de Código (10% dos pontos)
- Participação de Todos os Integrantes do Grupo (10% dos pontos)

Observe que a implementação de testes que demonstrem a corretude da sua solução é obrigatória.

A participação de todos dos integrantes do grupo no trabalho será validada caso todos os membros tenham realizado ao menos um *commit* relevante na árvore de fontes e/ou atuado na gestão do projeto, de forma importante (*ie.* criação de *cards*, bugs, *pull requests*, *merges*).

Discussões entre grupos da turma são encorajadas. No entanto, qualquer identificação de cópia do trabalho, total ou parcial, implicará na avaliação em zero, para ambas as partes.

6. Distribuição de Pontos Extras

Os grupos que optarem, podem realizar uma ou mais das atividades seguintes para obtenção de pontos extras nesse trabalho:

- Esboçar um diagrama de classes do projeto usando uma ferramenta de UML (1 ponto).
- Integrar a compilação do projeto com um ambiente de integração contínuo, como Github Actions, Jenkins ou TravisCI (1 ponto).
- Automatizar a compilação do projeto usando o sistema `make` ou `cmake` (2 pontos).
- Realizar a avaliação de desempenho da sua solução. (3 pontos)