TRABALHO PRÁTICO 1
IBD 2021/1

Sergio Mergen

A classe Table e o seu índice

- A classe Table está associada a um índice

- Função principal do índice
  - auxiliar na busca **em memória** por registros a partir de um parâmetro de filtragem sobre a chave primária

- Exemplos
  - Recuperar o registro que tenha a chave primária 200
    - Record rec = table.getRecord(200L);
  - Recuperar os registros que tenham a chave primária maior do que 200
    - List<Record> list = table.getRecords(200L, ComparisonTypes.GREATER_THAN);

A classe Table e o seu índice

- O índice dentro da classe Table é uma estrutura que se preocupa apenas com os aspectos referentes ao acesso em memória

- Detalhes de armazenamento físico ficam sob responsabilidade de outro módulo que se comunica com esse índice
  - O método **initLoad**() da classe Table faz a carga inicial das entradas do índice para a estrutura em memória
  - Todas as modificações em registros causam atualização do índice na memória
  - O método **flushDB**() faz com que todas as entradas do índice sejam salvas no arquivo

Entradas do índice

- Para cada registro, o índice deve guardar uma entrada
  - Essa entrada contém uma associação entre a chave primária do registro e região física onde o registro está localizado
  - A região física é determinada pela combinação entre blockId e recordId
- Uma entrada é armazenada na forma de um IndexRecord

```
public class IndexRecord {

    long recordId;
    long blockId;
    private long primaryKey;
    .
    .
    .
}
```

Interface Index

- O índice deve ser uma extensão da classe Index

```
public interface Index {

    public void clear();
    public void addEntry(Long blockId, Long recordId, Long primaryKey);
    public void removeEntry(Long primaryKey);
    public IndexRecord getEntry(Long primaryKey);
    public List<IndexRecord> getEntries(Long primaryKey, int comparisonType);
    public int getRecordsAmount();
    public Iterator iterator();
}
```

Funções da Interface Index

- public void addEntry(Long blockId, Long recordId, Long primaryKey);
  - Adiciona uma entrada no índice através da criação de um IndexRecord

- Uso
  - É usado quando um novo registro é criado
  - Também é usado quando todo o índice é carregado do arquivo pelo método **initLoad**()

Funções da Interface Index

- public void removeEntry(Long primaryKey);
  - Remove a entrada do índice

- Uso
  - É usado quando um registro é removido

Funções da Interface Index

- public IndexRecord getEntry(Long primaryKey);
  - Recupera uma entrada que esteja associada a uma chave primária específica
  - A entrada é recuperada na forma de um IndexRecord

- Uso
  - Para recuperar um registro que deve ser removido ou modificado
  - Para verificar se já existe um registro com aquele valor de chave durante uma inserção
  - Para satisfazer buscas iniciadas pelo usuário
  - Para satisfazer buscas iniciadas pelo organizador de registros da tabela

Funções da Interface Index

- public List<IndexRecord> getEntries(Long primaryKey, int comparisonType);
  - Recupera uma lista de entradas que satisfaçam os parâmetros de busca definidos
  - A lista é composta por instâncias da classe IndexRecord
- Valores possíveis para comparisonType:
  - EQUAL, DIFF, GREATER_THAN, GREATER_EQUAL_THAN, LOWER_THAN, LOWER_EQUAL_THAN;
- Uso
  - Usado para satisfazer buscas iniciadas pelo usuário

Funções da Interface Index

- public void clear();
  - Limpa a estrutura interna que armazena os IndexRecords

- Uso
  - Usado antes que os dados básicos da tabela sejam carregados pelo **initLoad**()

Funções da Interface Index

- public int getRecordsAmount();
  - Retorna a quantidade de IndexRecords armazenada

- Uso
  - O valor é usado para gravação do índice no arquivo
  - Também pode ser solicitado pelo usuário para saber a quantidade de registros presentes na tabela

Funções da Interface Index

- public Iterator iterator();
  - Devolve um Iterador que permite acesso sequencial pela lista de IndexRecords

- Uso
  - Usado para armazenar as entradas do índice no arquivo
  - Também pode ser usado pelo usuário para varrer todas as entradas presentes no índice

Instanciação de um índice

- Dentro da classe Table é feita a definição de qual tipo de índice deve ser usado
  - Atualmente, apenas um tipo foi implementado (Índice Hash)

```
public abstract class Table {

    .
    .
    .
    Index index = new HashIndex();
    .
    .
    .
}
```

Instanciação de um índice

- Para ser suportado, um tipo de índice deve estender a classe Index e implementar todos os seus métodos

```
public class HashIndex implements Index,  Iterable {

    public Hashtable<Long, IndexRecord> index = new Hashtable();

    @Override
    public void clear() {
        index.clear();
    }

    .
    .
    .

}
```

Trabalho Prático

- O índice hash implementado não suporta buscas por intervalo

```
public class HashIndex implements Index,  Iterable {

    public Hashtable<Long, IndexRecord> index = new Hashtable();


    .
    .
    .
    @Override
    public List<IndexRecord> getEntries(Long primaryKey, int comparisonType) {
        throw new UnsupportedOperationException();
    }
    .
    .
    .
}
```

Trabalho Prático

Objetivo

- Criar uma novo tipo de índice que também suporte buscas por intervalo
- Como?
  - Criando uma classe chamada xxxIndex, onde xxx é o nome do aluno
  - A classe deve estender a classe Index e implementar todas as funções herdadas
    - Inclusive o método getEntries(), que não é implementado pelo HashIndex

Trabalho Prático

- Funções que devem ser implementadas

```
public interface Index {

    public void clear();
    public void addEntry(Long blockId, Long recordId, Long primaryKey);
    public void removeEntry(Long primaryKey);
    public IndexRecord getEntry(Long primaryKey);
    public List<IndexRecord> getEntries(Long primaryKey, int comparisonType);
    public int getRecordsAmount();
    public Iterator iterator();
}
```

Orientações Gerais

- O índice usado pela classe Table é uma estrutura projetada para funcionar inteiramente em memória
  - Detalhes referentes à leitura e gravação no disco ficam em u módulo à parte

- Analise a implementação da classe HashIndex para ter uma noção de como as funções podem ser implementadas

Orientações Gerais

- Qualquer estrutura de dados pode ser usada/criada/aproveitada para fazer o gerenciamento das entradas do índice
  - Listas encadeadas, árvores, vetores, ...

- Obs. O uso de estruturas de dados não otimizadas pode acarretar em descontos na nota
  - Ex1. Usar uma lista encadeada não ordenada, que apresenta alto custo para realizar qualquer operação.

  - Ex2: usar duas estruturas de dados redundantes em vez de uma só

Orientações Gerais - Iterator

- A função iterator() devolve um Iterator criado especificamente para recuperar entradas do índice

```
public Iterator iterator();
```

- Existem várias formas de devolver um iterador
  - Usar uma estrutura de dados do próprio Java para o gerenciamento das entradas do índice e devolver o resultado da chamada à função iterator() existente
  - Criar uma classe que estenda Iterator e devolver uma instância dessa classe

Orientações Gerais - Iterator

- usar uma estrutura de dados do próprio Java para o gerenciamento das entradas do índice e devolver o resultado da chamada à função iterator() existente

```
public class HashIndex implements Index,  Iterable {

    public Hashtable<Long, IndexRecord> index = new Hashtable();

    .
    .
    .

    @Override
    public Iterator iterator() {
        return index.values().iterator();
    }
```

Orientações Gerais - Iterator

- Criar uma classe que estenda Iterator e devolver uma instância dessa classe

```
public Iterator iterator() {
        return new HashIndexIterator(this);
    }
```

```
class HashIndexIterator implements Iterator{
        Iterator it;
        public HashIndexIterator(HashIndex hashIndex){
            this.it = hashIndex.index.values().iterator();
        }
       @Override
        public boolean hasNext() {
            return it.hasNext();
        }
       @Override
        public Object next() {
            return it.next();
        }
}
```

Orientações Gerais - Iterator

- Existem várias possibilidades para implementar uma classe Iterator
  - Usar uma estrutura de dados do próprio Java para o gerenciamento das entradas do índice e reaproveitar o iterador que essa estrutura disponibilize
    - Estratégia usada pelo HashIndex
    - Poderia ser feita de forma direta, sem a criação de uma nova classe Iterador
  - Usar uma estrutura de dados do próprio Java temporariamente para armazenar todo o conteúdo do índice e reaproveitar o iterador que essa estrutura disponibilize
    - Método simples, mas menos eficiente
  - Implementar as funções de iteração na mão

Avaliação

Método/função	nota
public void clear()	10%
public void addEntry(Long blockId, Long recordId, Long primaryKey);	10%
public void removeEntry(Long primaryKey);	20%
public IndexRecord getEntry(Long primaryKey);	10%
public List<IndexRecord> getEntries(Long primaryKey, int comparisonType);	20%
public int getRecordsAmount();	10%
public Iterator iterator();	20%

Testes

- Para testar, use as funções disponibilizadas pela classe Table
  - Record getRecord(Long primaryKey) ;
  - getRecords(Long primaryKey, int comparisonType);
  - Record addRecord(long primaryKey, String content);
  - Record updateRecord(long primaryKey, String content);
  - Record removeRecord(long primaryKey);
  - void flushDB();
  - void createTable();
  - int getRecordsAmount();
- A classe Main já demonstra algumas formas de usar essas funções

Entrega

- Entrega pelo moodle
  - Prazo final: 20/06 às 21:00
  - A cada 24 horas de atraso, a nota é decrementada em 50%.

- Não entregue o projeto inteiro
  - Apenas a classe que estende Index.
  - Nome da classe: xxxIndex, onde xxx é o nome do aluno
  - A classe deve pertencer ao pacote ibd.index
  - Se outras classes forem necessárias, defina-as como privadas, dentro da classe principal

- O trabalho é **individual**
  - O compartilhamento de código entre alunos leva à anulação da nota