MAC-211 — Laboratório de Programação

Exercício Programa 4: JBase - Gerenciador de Índices

Entrega: 01 de julho de 1999

Nesta etapa final do projeto o objetivo é implementar um gerenciador de índices que permita a execução eficiente das operações `exactMatchQuery` e `rangeQuery`.

1 O Que Muda na API do JBase

Em termos da API do JBase, vamos adicionar um novo método à interface `RecordSet`:

```
public interface RecordSet {

    public abstract void createIndex(String keyName)
            throws java.lang.NoSuchFieldException, NotSupportedException,
                   NotEnoughSpaceException, java.io.IOException;

    ... // outros métodos como no EP3

}
```

O método `createIndex` cria um índice para uma coleção de registros. O parâmetro `keyName` é o nome do campo sobre o qual o índice será criado. O trecho de programa abaixo cria uma coleção de registros de alunos e em seguida cria um índice sobre o campo `numUSP` dos registros dessa coleção:

```
// cria coleção de registros
RecordSet alunos = db.createRecordSet("Alunos", Aluno.class);

// cria índice sobre o numUSP
alunos.createIndex("numUSP"); // numUSP é um campo da classe Aluno
```

Consultas envolvendo um campo indexado são processadas eficientemente pelo JBase. Considere, por exemplo, a seguinte consulta:

```
DBIterator it = alunos.exactMatchQuery("numUSP", new Integer(2020202));
Aluno a = new Aluno();
while (it.next(a) != DBIterator.END) {
    System.out.println(a.nome);
}
```

Se o campo `numUSP` não fosse indexado, o JBase precisaria examinar todos os registros de alunos e “filtrar” os que não satisfazem a condição do `exactMatchQuery`. Mas, como temos um índice sobre o campo `numUSP`, o JBase usa o índice para localizar os registros de alunos com número USP igual a 2020202. (É de se esperar que não exista mais de um aluno satisfazendo essa condição.)

2 O Que Deve Ser Implementado

Os índices devem ser implementados como árvores-B+. Para uma boa discussão sobre organização de índices e sobre árvores-B+, veja os textos "File Organizations and Indexes" e "Tree-Structured Indexing", disponíveis no xerox do CAMAT. Antes de prosseguir com o projeto, leia a seção 4.6 do primeiro desses textos e as seções 5.2–5.5 e 5.8.3 do segundo.

Cada índice é uma árvore-B+ armazenada em disco. Cada nó dessas árvores é uma página do disco. Os nós folhas contém entradas[1] da forma *(k,rid)*, onde k é um valor do campo sobre o qual o índice está definido e *rid* é o record id de um registro que tem o valor k nesse campo.

Tendo sido criado um índice para um `recordSet`, as operações de inserção e remoção de registros de dados devem atualizar também o índice. Consultas envolvendo campos não indexados são executadas como no EP3. Consultas envolvendo campos indexados fazem uma busca na árvore-B+.

A adição de um registro a um `recordSet` com um índice causa a adição de uma entrada de dados a um nó folha do índice. Para adicionar essa entrada, use o algoritmo de inserção de entrada numa árvore-B+. Em caso de sobrecarga de um nó (em qualquer nível da árvore), esse algoritmo faz a divisão do nó sobrecarregado[2].

A remoção de um registro de um `recordSet` com um índice causa a remoção de uma entrada de dados de um nó folha do índice. Neste EP você pode simplesmente remover a entrada de dados do nó folha, mesmo que esse nó fique sub-ocupado. Não precisa fazer a redistribuição das entradas ou a junção de um nó sub-ocupado com um nó irmão adjacente.

3 Simplificações Sugeridas

Para viabilizar esta etapa do projeto você pode fazer as seguintes simplificações:

1. Uma "versão profissional" do JBase deveria permitir a criação de múltiplos índices para o mesmo `recordSet`. (Exemplificando: a coleção de registros de alunos poderia ter um índice sobre o campo `numUSP` e outro sobre o campo `nome`.) Para simplificar o projeto, entretanto, você pode implementar um gerenciador de índices que crie no máximo um índice por `recordSet` e lance a exceção `NotSupportedException` caso o método `createIndex` seja chamado para criar um segundo índice para um `recordSet` já indexado.

2. Numa "versão profissional" do JBase, o conjunto de tipos indexáveis incluiria os tipos primitivos de Java (exceto `boolean`), o tipo `String` e possivelmente outros tipos (`java.util.Date`, por exemplo). Mas, para simplificar o projeto, você pode restringir o conjunto de tipos indexáveis aos tipos `int` e `String`. Faça o método `createIndex` lançar a exceção `NotSupportedException` caso ele seja chamado para criar um índice sobre um campo cujo tipo não é indexável.

3. Uma "versão profissional" do JBase permitiria a criação de índices sobre campos não determinantes, mas isso não é essencial neste EP. Você pode implementar um gerencia-

[1] Os textos no xerox do CAMAT chamam de entradas de dados (*data entries)* as entradas contidas em nós folhas e de entradas de índice (*index entries*) as contidas em nós não folhas.

[2] Algumas variações do algoritmo de inserção procuram evitar a divisão de um nó sobrecarregado redistribuindo as entradas desse nó com um nó irmão adjacente. Você não precisa fazer isso neste EP.

dor de índices voltado para campos determinantes[3]. Uma vez criado um índice sobre um campo, sua implementação do JBase pode considerar que o campo é determinante e rejeitar a adição de um registro cujo valor nesse campo já aparece no `RecordSet`. Se você optar por esta simplificação, faça o método `add` da interface `RecordSet` lançar a exceção `NotSupportedException` em caso de tentativa de adição de registro com campo indexado igual ao de um registro já existente no `RecordSet`.

Essas simplificações são opcionais (se você tiver gás para uma versão não simplificada, melhor!) e tem o objetivo de reduzir o trabalho envolvido nesta etapa do projeto.

4 Sobre a Implementação

- O formato dos registros do diretório de arquivos de um `Database` deve mudar nesta etapa, pois agora o diretório de arquivos precisa manter informações sobre os índices existentes. Com a simplificação 1, basta incluir em cada registro campos que informem se existe ou não um índice para o arquivo, qual o campo indexado e qual a página raiz do índice.

- As páginas de índice podem ser armazenadas usando o formato `HFPage`. As páginas folhas contém entradas da forma *(k,rid)*, como vimos na seção 2. As páginas não folhas contém entradas da forma *(k,pid)*, onde *k* é um valor separador (um possível valor do campo sobre o qual o índice está definido) e *pid* é o page id de uma página filha. Note que o formato das entradas é o mesmo nos dois casos: tanto o *rid* como o *pid* são do tipo `long`.

- É importante distinguir as páginas folhas das não folhas, pois os algoritmos de busca e inserção as tratam de modo diferente. Use o campo "page type" da `HFPage` para identificar o tipo da página.

- As páginas folhas formam uma lista duplamente ligada. Use os campos "previous" e "next" da `HFPage` para implementar essa lista. As páginas não folhas não estão numa lista duplamente ligada, mas tem um ponteiro (page id) a mais, para a primeira página filha. Faça o campo "previous" apontar para a primeira página filha.

5 O Teste desta Etapa

Modifique o(s) programa(s) de aplicação que você escreveu na etapa anterior, de modo que um ou mais campos sejam opcionalmente indexados. Faça a aplicação criar um `recordSet` indexado e povoá-lo com um número de registros suficientemente grande, de modo que a altura da árvore-B+ que implementa o índice seja pelo menos igual a 1. Compare o desempenho das consultas a esse `recordSet` com o das consultas a um `recordSet` equivalente, porém não indexado.

[3] Um campo é determinante quando não for permitida a existência de mais de um registro com o mesmo valor no campo. Pensando em nosso banco de dados de alunos, o `numUSP` seria um exemplo de campo determinante. (Estamos assumindo que não existam dois alunos com o mesmo número USP.) Um campo `rua` seria não determinante, pois podem haver dois alunos morando na mesma rua.

6 O Que Voce Deve Entregar

- Pelo sistema de entrega eletrônica de trabalhos: um arquivo `.tar` contendo:
  - todos arquivos fonte (`.java`) desta etapa;
  - um `Makefile` que compila esses arquivos e roda o(s) programa(s) de aplicação.;
  - um arquivo `README` com instruções adicionais que você julgar necessárias. Provavelmente será o caso de incluir instruções de uso do(s) programa(s) de aplicação.

  Os ítens acima devem ser "empacotados" num único arquivo `.tar`, pois o sistema de entrega eletrônica recebe só *um* arquivo. Caso o arquivo `.tar` fique muito grande, comprima-o com o `gzip` e entregue um arquivo `.tar.gz`.

- Em papel: listagens dos arquivos `.java`, do `Makefile` e do arquivo `README`.

Uma pessoa que recebeu o seu arquivo `.tar` (ou `.tar.gz`) e o desempacotou deve poder compilar os fontes e rodar os testes com *muita* facilidade. Use o `Makefile` para automatizar ao máximo esse processo. Caso sejam necessários diretórios organizados de modo especial, inclua tais diretórios no arquivo `.tar`. Se for preciso acertar alguma variável do environment antes de rodar o `make`, não deixe de colocar essa informação no arquivo `README`.

Atenção: o sistema de entrega eletrônica aceita arquivos de no máximo 200 KB. Antes de empacotar o seu diretório de trabalho, remova os arquivos desnecessários, especialmente os grandes (arquivos com databases de teste, por exemplo).

Bom Projeto!