Lab 3: NEUStore

Due date: Tuesday, April 15, end of lab period.

Lab Assignment

The purpose of this lab is to get acquainted with NEUStore, the Java package for buffer management and paginated disk access. This is a team lab, to be performed by the project teams.

NEUStore

NEUStore is a Java package providing descriptions of a number of abstract classes for creation and maintenance of paginated index structures on disk. In addition, NEUStore provides sample working implementations of the most important classes. In particular, NEUStore contains the following.

- **Buffer Manager.** NEUSTore offers an abstract class `DBBuffer` representing the DBMS buffer. Buffer management policies (the asynchronous part of the buffer manager) are represented by the abstract methods of this class, which need to be instantiated for each type of buffer manager. In addition `DBBuffer` offers the read/write/pin/unpin API (the synchronous part of the buffer manager) which allows application programs to communicate with the buffer to read, write and pin/unpin pages.

  NEUStore also offers class `LRUBuffer` which extends `DBBuffer` and implements the traditional Least Recently Used buffer management policy. This lab assignment, as well as the course project will assume that you use `LRUBuffer` as your buffer manager.

- **Disk Pages.** NEUSTore offers an abstract class `DBPage` representing a single disk/buffer page. It also offers two classes, `NaiveHeapFilePage` and `HeapFilePage` which extend `DBPage` and implement a simple heap file page (storing only integers) and a generic heap file page respectively.

This lab assignment requires each team to study the source code of the implementations of `NaiveHeapFilePage` and/or `HeapFilePage` and modify it as needed to meet the needs of the lab.

- **Index Files.** Collections of disk pages from *index files.* NEUStore provides an abstract class `DBIndex` representing an index file on disk. Index files can contain disk pages of different types. NEUStore includes two classes, `NaiveHeapFile` and `HeapFile` which extend `DBIndex` for the naive and generic heap file implementations.
- **Tests and examples.** NEUStore comes with two examples mentioned above. The first example is an implementation of a naive heap file, which only stores single integers on disk pages. The second example is an implementation of a generic heap file, which can store any *(Key, Value)* records (where both the key and the value may be compound). In addition, `NEUSTore` provides a sample test program showing how the generic heap file can be initialized, populated and queried.

Assignment

Using NEUStore as the back end, create a program that does the following.

R1. **Dataset description.** Your program shall provide heap file storage for the data found in the list.txt file used in CSC 365. The URL of the file is:

http://users.csc.calpoly.edu/ dekhtyar/365-Spring2008/labs/lab1/list.txt

The file stores information about students of an elementary school. The file format is:

```
StLastName, StFirstName, Grade, Classroom, BusRoute
```

Here, `StLastName` and `StFirstName` are the last and the first name of the student (in ALL CAPS), `Grade` is the grade the student attends (0 means the student attends kindergarden), `Classroom` is the classroom the student attends and `BusRoute` is the number of the school bus route the the student takes to get to school (0 means the student is no taking any school bus).

A sample line from `list.txt` is

```
DEMARTINI, DEWAYNE, 6, 102, 55
```

("Dewayne DeMartini is a 6th grade student assigned to classroom 102, who takes route 55 school bus to school.")

The file contains 60 records.

R2. **Format assumptions.** Your program needs only to provide storage for the data found in `list.txt`. You can (and shall) determine the lengths of the string fields in this file based on its contents, but there is no need to expect longer first or last names.

R3. **Disk Storage.** You shall design appropriate format for storing records from the `list.txt` file. You shall design appropriate disk page format (including page headers). You shall extend existing NEUStore implementation[1] to create appropriate classes for each disk page format you intend to use in the program[2], as well as for the index file in which all data will be stored on disk.

R4. **Internal API.** Your program shall provide implementations[3] of the following methods for the index file:

- `insert` : insert a given record into the index structure;
- `delete` : delete a record from the index structure;
- `update` : replace a specified record with a new one;

(you shall make decisions on the arguments for these methods).

R5. **Program flow.** Your program shall work as follows:

- When the program is started, it reads and parses the contents of `list.txt` file. You can assume that the file is in the working directory of the program.
- The program then builds the index structure(s) for storing the contents of the file, and inserts, one-by-one all records from the file into the index structure.
- The program scans the index structure and outputs all records stored in it, in the order in which the records are accessed.
- The program scans the index structure and outputs records of all students who go to 5th grade.
- The program deletes all records of students whose bus route is 0.
- The program scans the index structure and outputs all records in it.
- The program reinserts one-by-one all deleted records. [4]
- The program scans the index structure and outputs all records in it.
- The program (retrieves and) updates the first record stored in the index and replaces the first name of the student with 'JUAN'.
- The program scans the index structure and outputs the first ten records in it.

[1] You may, if you want, write all your code from scratch, but using NEUStore-provided classes will probably be easier.

[2] Most likely, only one type of disk page, outside of the file header page is needed for this application.

[3] You are allowed, where suitable to rely on NEUStore implementations

[4] Feel free to "cheat" here. You can extract all records with bus route 0 into a separate file, or create a separate global data structure, to make identification of records easier.

Notes and Comments

The data structure you need to implement is more complex than the `NaiveHeapFilePage` class, but it is simpler (because it is fixed) than the `HeapFilePage` class. In fact, it can be done as an instance of using `HeapFilePage`, but I recommend modifying one of the two classes to create a data structure for exactly the record structure you need.

In addition to the insert/delete/update API, the program essentially requires you to implement a scan operation: starting from the first index page, read and output every single record (or read and output every single record up to a given number of records). In addition, you need a simple *selection* operation, a scan of the index file, retrieving and outputing only the records of 5th graders.

The delete procedure described in **R5.** can be performed as follows: your program employs a scan of the index structure, with a check of the bus route value (this is a simple modification of the "select 5th-graders" scan), then the API delete method is issued for each discovered record.

Submission Instructions

Your work essentially modifies the NEUStore package. You shall submit the entire package: both the new files you create, and the original NEUStore distribution files. In the root directory of your submission, put a plain text README file containing the following information:

- Group name.
- Group members, emails.
- Compilation, running instructions.
- The name of the data file you are creating. (I will run your program, and then will manually check the data file).
- A brief summary of the group's approach to the project. List any major design decisions (which classes were extended, how much code was borrowed, how much is new). Also, evaluate how hard/easy it was for your group to understand how to use NEUStore. List any outstanding questions about the functionality of NEUStore you group has.

Submit all your files in a single archive. Accepted formats are `gzipped tar` (`.tar.gz`) or `zip` (`.zip`). Name the file `lab3-<groupname>.ext`, where `<groupname>` is the name of your group.

Email your archive to *dekhtyar@csc.calpoly.edu*.