Concordia University
Department of Engineering and Computer Science

Comp 6521: Advanced Database Technologies

Programming Report

Dapeng Wang
Kaichen Zhang
Nour Yaghmour

Description of the program:

Technical implementation details:

As the operation of Java I/O needs the interaction of main memory and disks, we introduce the concept of stream. The file size is extremely large so we need to use buffers to decrease disk I/O times.

Because the records of the file are consecutive 100bytes, we need to extract the information of ages as the attribute to produce the secondary index. So in order to build our database system, our program need to read the input file stream as byte array: byte [] of 4KB size (1 block) and extract the ages found and locate their addresses. On the other hand, we need to create 1 bucket for each age to store the addresses that belong it. In order to decrease the I/O of output, we store the index information in the output buffer and organize it.

The general structure of the code (main program and the sub-programs) :

Our program is divided in to three main parts: (r my points correct?)

1-Reading the file: create an input buffer that reads the input file one block by one block and load it to the main memory in order to extract all the ages' information we need for building the index.

2-Building the index: Create a list of a list of size 100 that contains a byte list for every age. Every time, we read the age from the input buffer, extract the location of the age and add it to the byte list specified for its age. Then, Output an index file for every list.

3- Answering Queries: When querying a specific age, the program goes to the index file, read the index address of the expected age, use the address to check the file one block by one block. If the age is found, store the tuple in a byte array(size of 1 block) and continue to the end of the file. Every time the byte array is full, we output the byte array to see the results.

The algorithm and storage structure:

1. Using binary search to search the sorted keys which consists of non-duplicate ages.

2. Store the addresses of a record in the list of the list of buckets file name.

```
List<List<Byte[]>> indexBucket = new ArrayList <List<Byte[]>>();
```

3. As the number of the addresses of records is 20billion, we need at least 5 bytes to store the address.

```
Byte[] addr = new Byte[5];
addr[0] = (byte) (recordNum2 & 0xff);
addr[1] = (byte) ((recordNum2 >> 8) & 0xff);
addr[2] = (byte) ((recordNum2 >> 16) & 0xff);
addr[3] = (byte) ((recordNum2 >> 24) & 0xff);
addr[4] = (byte) ((recordNum2 >> 32) & 0xff);
```

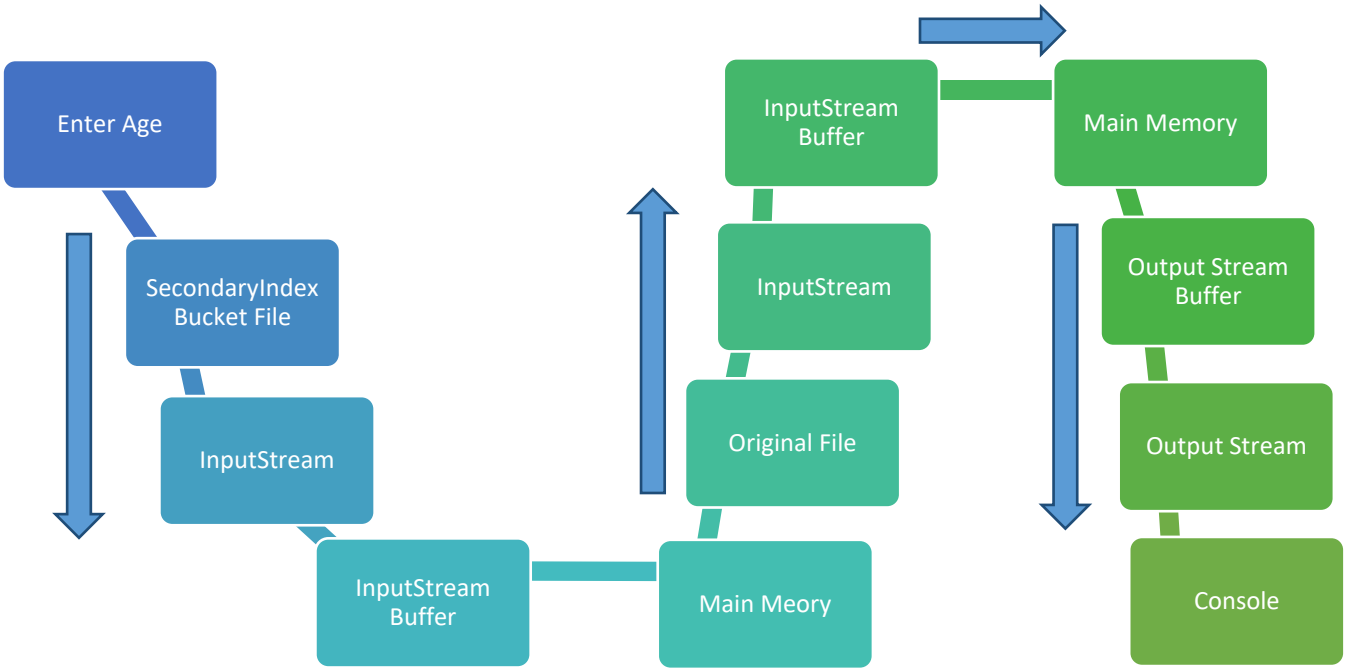
4. The input stream buffer will load again while the main memory has processed the buffer. The function fill() will check the current pointer and the remaining available buffer size, if available buffer size is null the buffer will load from input stream.

5. The output stream buffer will only output the largest size of index address and output enough data to keep space to load the next production of index address.

Accessing data:

Index Creation:

Age query:



Description of classes :

1- IndexCreation Class:

This class is responsible for reading 1 block of input buffer from original file and creating index list that contains the extracted ages and their location in the original file.

inputStreamRead() function **:**

- Create a list of a list of type byte called *index*.
- Create a byte array e of size 100(for all the 89 ages) and add them to *index.*
- Initialize *recordNum* variable to store the record number of each age.
- Using the input Stream buffer, Read 1 byte array block (4000 byte) of the original file, read the bytes 39 and 40 of each record to extract the age and increase the *recordNum* value.
- outputReamainedIndex: output the remained index(that means after dealing with the whole input, there may be some index haven't been output)

addIndex(age,record) function:

Check indexCapacity,

if there is a space:

Get the age list needed from the index list

Add the new address

Set the age list back to the index list

If the amount of index for one age is larger than 819,(819*5=4095<4096) we output the index for that age.

2- Query Class:

This class is responsible for answering queries and outputting the result

QueryByAge(age) function **:**

- Create two input stream buffers (byte array) for reading both from created index file(*indexblock*) and from the input file (*datablock*).
- Transform the address from byte to long type and search for it in all data blocks. When the address is found, read the 100 byte tuples for the needed age and store them in the byte array (*record*).
- Finally, every time the byte array is full, we output the results (result.txt).

Difficulties Faced:

Limitation of implementation ….please combine here ignoring our names

Kaichen Zhang:

1. How to handle the Java I/O, for the first time we are not clear about how the parts are connected. So when I tried to create the index I just use input stream to read the entire file. But it really costs.

2. Do not understand the buffer operation, the objects between buffer and input stream.

3. The problem of assign the address to index with fixed length. This problem was fixed when I understand the total operation is based on byte array, so I just assign them using a binary way.

4. Each buffer read and write is considered one I/O, but I don't understand the output buffer I/O counting.

Nour Yaghmour:

1. Decreasing I/O times while limiting the buffer to 1 data block per read.

2. Difficulties in understanding the byte type and its transformations.

3. Some problems in understanding the buffer streaming idea, but it was solved when merging the program and testing it.

4. Difficulties in understanding some functions in the program.

Dapeng Wang: ….

Division of Tasks:

Kaichen Zhang:

Creating the index file task + writing Report….

Dapeng Wang:

Building the index +handling I/Os+ merging the whole program……

Nour Yaghmour:

Answering Queries task (was changed by Dapeng during the merging process) + writing the report.

Result, analytics, and comments:

Configuration 1:

For testing an input file of (3.8million tuples) with a 5MB main memory

Time for building the index file : 27326ms

Number of I/Os : 99677

Time for querying a specific age : 586ms

Number of I/Os: 96272

Number of records we get: 48640

Configuration 2:

For testing an input file of (3.8million tuples) with a 2MB main memory

Time for building the index file : 27918ms

Number of I/Os : 99677

Time for querying a specific age : 428ms

Number of I/Os: 96272

Number of records we get: 48640

Average income for all age groups:

age18~19:95102

age20~29:97947

age30~39:97282

age40~49:96469

age50~59:97646

age60~69:96826

age70~79:97179

age80~89:97441

age90~99:94675