Database Systems

Assignment 1

Daniel Atanasovski

S3658474

Contents

[Task 1 – Heap File 3](#_Toc100046331)

[Pre-processing 3](#_Toc100046332)

[Java Running 3](#_Toc100046333)

[Java Implementation 3](#_Toc100046334)

[Results 4](#_Toc100046335)

[Task 2 – Range Query 5](#_Toc100046336)

[Java Implementation 5](#_Toc100046337)

[Results 5](#_Toc100046338)

# Task 1 – Heap File

## Pre-processing

The pre-processing script is implemented in python and depends on the package ‘Pandas’. It can be installed with the following command:

*Sudo pip install pandas*

The script simply removes un-needed columns within the source data file, artist.csv, and outputs it into the destination file ‘artist\_processed.csv’. The naming of the source data file is expected to be ‘artist.csv’ by the script and requires manual altercations to the python script to change the expected filename. The script can be run with the following command:

*Python preprocess.py*

## Java Running

After unzipping the source zip file, the java code can be compiled with the following command (assuming all java source files are in the src/ directory):

*Javac src/\*.java*

After compiling the code, the program expects the arguments ‘-p’ followed by the heap file size in bytes and then the source csv file that will supply the data to the program. An example command is as follows:

*Java -cp src dbload -p 4096 artist\_processed.csv*

## Java Implementation

The program first confirms the positions of the columns and records those index positions so they can be referred to later when pulling that data. Removal of header rows is now removed in the cleaning process. Records are generated by reading each line of a CSV, converting each datapoint to their respective types, with dates being the days since the epoch (1970-1-1) as it is the standard and fits the use case of the data.

Records are padded to the maximum length of the field with null-terminated characters. In this project, the largest value witnessed in the dataset determines the maximum size of the field, for example, the name field has a maximum size of 70 bytes. In the case of dates, they are converted to long values represented as days since the epoch as they provide a clean interaction with the Java API. If the wikipageID or date values are null or invalid, they are set to the maximum value of their datatype.

If data is invalid, for example, a date range is given in the birthdate column, the data is cleaned to ensure the data is valid, and in this case, would set the birthdate value to the maximum value.

Since the record size is fixed, we can determine the number of records per page, and with that generate a byte array of the page size that is then added to by records that are assigned to that page.

## Results

Some example output posted from an AWS instance, showcasing an average time of 9 seconds to generate a Heap File from a processed CSV.

Text

Description automatically generated

To Validate the data, we will take the first result of the processed data and convert some data types from the Heap File for comparison.

Text

Description automatically generated

Using a program like HxD we can view the byte data of the Heap File. The first 8 bytes are the values surrounding the birthdate, followed by 8 bytes for the death date, then the integer value for the wikipageID with 4 bytes before the string values. The first 8 bytes come to a value of 1371 which corresponds to the days since the epoch and is accurate to the value of the first record’s birthdate, 3/10/1973.

# Task 2 – Range Query

## Java Implementation

The range query is simple and begins by validating the input from the arguments, including whether the dates are valid and whether the Heap File is a valid file. After parsing the page size from the Heap File argument, we can determine the bytes to be read on each input of the Heap File.

The date arguments are parsed to the LocalDate datatype as it provides a clean API to convert from the days since the epoch (1970-1-1) to the LocalDate object, which is important to perform range query operations on the data.

On reading a page size worth of data from the Heap File, the data is split according to the fixed constant of record size, which is known from task 1. The record size of data is converted to an object that performs validation on the input and converts to the correct data types. After the conversion to a record object, the date is compared to the two-argument dates, in which case if it exists between the two provided dates, the name field, and birth date are printed out to the output, otherwise, it is ignored.

This is repeated until all pages are read and each record on the page is compared to the input dates. It finally outputs the number of matching records for the query.

## Results

An example query/command to run the program is provided:

*Java -cp src birthdate heap.4458 19700101 19701230*

This query should provide a lengthy output of 931 records individually printed to the console with the name and birthdate field, overall taking 8.26 seconds to complete the query.

Text

Description automatically generated

Output of 2 records per page with printing individual results

This query was tested on a pagesize of 4458 bytes, which fits two records perfectly. Testing the query on a larger pagesize, provides no significant performance boost to the query.

Text

Description automatically generated

Output of 4 records per page without printing individual results

Text

Description automatically generated

Output of 4 records per page with printing individual results

However, outputting every result as they are found in the Heap File adds a small performance cost to the query of an average 0.2 seconds, and should be kept in mind for when the results are quite large.

Escalating the query to a larger difference of dates results in a similar query time to the smaller date arguments.

Text

Description automatically generated

Output of 4 records per page with the printing of individual results with a query of dates between 19000101 and 20001230

Text

Description automatically generated

Output of 2 records per page with the printing of individual results with a query of dates between 19000101 and 20001230

As can be seen increasing the records per page has a very minor performance boost despite the query returning over 50,000 results.

Text

Description automatically generated

Output of 4 records per page without the printing of individual results with a query of dates between 19000101 and 20001230

The performance difference is growing significant between no output for individual results and output, with a query of 50,000 results resulting in 0.44 seconds difference, which is a 5% performance boost for this query size. As the results from the query escalate it is worth considering the impact of printing each individual result to the standard output, and worth exploring streaming to output files.