CMPUT 291 Fall 2019
MINI PROJECT II
DESIGN REPORT

Group Members:

1. Vanika Dhamija - vanika (1503248)
2. Jordan Los - los (1550572)
3. Evan Timms - perretti (1504825)

(A) A General Overview of the System with a Small User Guide

Mini Project II of CMPUT 291 aims at teaching the concept of working with data in the physical layer. This is done by building an information retrieval system, using the Berkeley DB library for operating on files and indices. To meet the requirements of this project, programs are written to keep data in files and maintain indices that provide basic searches over data. Another goal of the project is to optimize the efficiency of the programs and queries in terms of running time.

The project is divided into three phases. Phase 1 involves preparing data files for constructing indices in phase 2. This is done for data that consists of a set of emails and. each email record consists of an id, date when the email is sent, a subject, a body, and the fields from, to, cc and bcc. The code for this part of the project is in the file *PrepareFiles.py* Phase 2 involves building indexes and this is done by sorting the files built in phase 1. The Linux *sort* command and Berkley DB's hash and B+-tree functionalities are used to do this. The code for this part of the project is in *LoadFiles.sh*. Finally, data retrieval is done in phase 3 of the project. The queries and required functions are written in the file *DBMS.py* and a program *main.py* is written to process queries based on the project description and user inputs.

The program can be run using terminal and typing in following commands for respective phases of the project:

- Phase 1: python3 PrepareFiles.py xml_file_name.xml
- Phase 2: sh LoadFiles.sh
- Phase 3: python3 Main.py

For Phase 3, DBMS.py file has the code for all the queries and functions which are called in Main.py to process the queries. On running Phase 3, the user will be prompted to follow or run commands to perform queries or exit the system.

(B) Description of Algorithm for Efficient Evaluation of Queries

Phase 1: Preparing Data Files

The given xml file is read and data is written to terms.txt, emails.txt, dates.txt and recs.txt based on given requirements. Four functions described below are defined which are called in main to get the required data in the above-mentioned files.

- write_to_terms(row, subject, body):
  This function writes to terms.txt file based on project description.
- write_to_emails(row, frm, to, cc, bcc):
  This function writes to emails.txt file based on project description.

- write_to_dates(row, date):
- This function writes to dates.txt file based on project description.
- write_to_recs(row, line):
  This function writes to recs.txt file based on project description.

The main function then iterates through the xml file and gets the required data between different xml tags.

Phase 2: Building Indexes

Sorted files, namely, terms.txt, emails.txt, dates.txt and recs.txt, are obtained as a result of running phase 1 files. These files create the following four indices:

1. A hash index on recs.txt with row ids as keys and the full email record as data
2. A B+-tree index on terms.txt with terms as keys and row ids as data
3. A B+-tree index on emails.txt with emails as keys row ids as data
4. A B+-tree index on dates.txt with dates as keys and row ids as data

This is done using the Berkeley DB and a bash script that uses the given perl script and performs db_load and db_dump functionality while preserving duplicates in the indexes and only preserving unique keys.

Phase 3: Data Retrieval

This phase of the project allows the user to open the databases and process queries. The default output format of each query is the row id and the subject field of all matching emails. The user can change the output format to full record by typing *"output=full"* and back to id and subject field only using *"output=brief"*.

Main.py

- Regular Expression Work:
  Term_query_with_prefix, term_query_without_prefix, email_prefix, email, date_prefix use regex to check for respective fields and performs matches when user input is captured.
- Main function:
  The main function in this file checks for the output mode (brief or full) and also checks for the user quitting the program. It also prompts the user to enter queries to be processed and then prints the result in the chosen output mode. For running the queries, the programs checks certain conditions to process the query.

DBMS.py

- runDateQuery: It takes date and operator as arguments and adds all row ids that match to mast_ids.
- runEmailQuery: It takes field and email as arguments and adds all row ids that match to mast_ids.
- runTermQuery: It takes field and term as arguments and adds all row ids that match to mast_ids.
- getResults: Prints results from the records that correspond to mast_ids in the specified output mode.
- resetQuery: It resets the master ids to prepare for a new query.
- addToMasterIds: It intersects a set of ids to the master copy held in memory.

(C) Testing Strategy

- Started building the program by having all the set-up data in place and writing the code for one phase at a time and testing it subsequently. This allowed to build a bug-free program in small steps.
- Code for regular expressions was also tested thoroughly to ensure it does not miss any cases or includes extra unexpected cases.

- All the code files from the three phases were combined to implement an executable program which is tested using the given data files. The program is also tested for scalability and efficiency.
- Finally, it was ensured that the project code works well with the python version on the lab machine and some print statements were added to tell the user about invalid query and end result.

(D) Group Work Break-Down Strategy

Break down of work items:

The group worked together on various parts and phases of the project for most of the time. The phase 1 was implemented by all three members together. Evan implemented Phase 2. The work for phase 3 was distributed among the members. Vanika did the initial setup which was refactored by Evan and Jordan for a more efficient implementation. Evan did runDateQuery, output mode and rest of the setup code for main. Vanika and Jordan did the other two queries. Jordan also worked on the regex part. Vanika prepared the design document and captured all the ideas. The group tested the final code together and prepared the final submission.

Estimate of time spent and progress made by each partner:

- Evan: 8 - 10 hours
- Jordan: 8 - 10 hours
- Vanika: 8- 10 hours

Method of coordination to keep the project on track:

- The group met in the beginning of the project to work together on Phase 1 and 2. The overall strategy of the project was also discussed and work for Phase 3 was divided among the group members.
- The group also met a day before the submission deadline to work on remaining queries of phase 3, implementing output mode functionality and fixing code conflicts and performing code refactor..
- The group also tested the entire project together on the submission day and also to finalize the submission.
- The code was pushed on GitHub so the members could see the current state of the project at all times and review each other's work as well.
- All the other communication was done in a group chat, including modifying work distribution as necessary and tracking progress.
- The design document was prepared by Vanika and was shared as a google doc for further collaboration.

(E) Assumptions made in the Project

No further assumptions were made in this assignment