CMPUT 291 - Mini Project 2 Design Document

1 Overview and User Guide

The following software package is comprised of 3 phases. Phase 1 takes email data from an XML file and outputs the data into four files: terms.txt, emails.txt, dates.txt, and recs.txt. Phase 2 sorts these files and builds four indexes. Phase 3 is responsible for data retrieval, and queries the data.

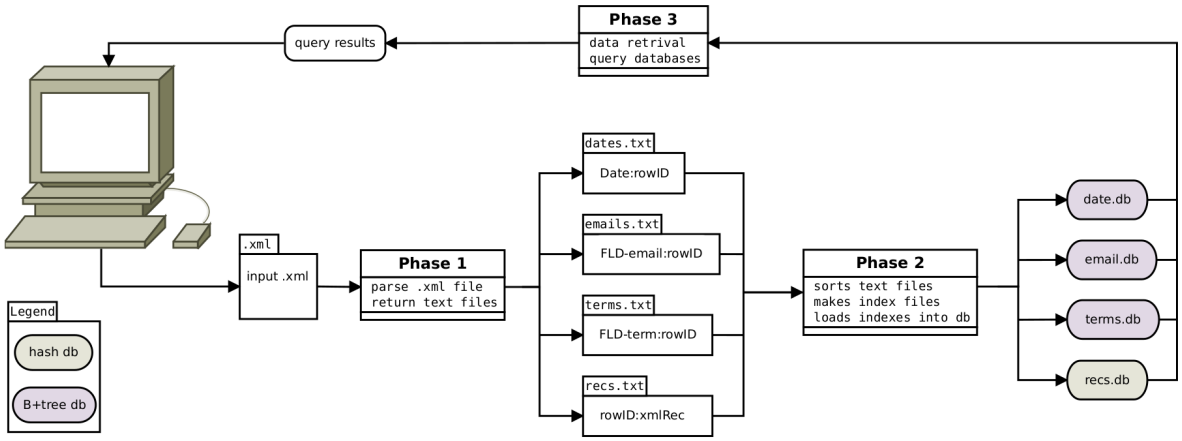

Figure 1: Flow diagram of files, data, and different phases of the software. For more implementation details see *Query Algorithms*.

1.1 Phase 1: Preparing Data Files

Preparation of the data files is completed by the python program `phase1.py`. The software reads in an `.xml` file specified by the user line by line, parses the line, and writes the data into four output files: `terms.txt`, `emails.txt`, `dates.txt`, and `recs.txt`. The phase 1 software can be run by entering the following command in the Linux terminal:

```
$ python3 phase1.py
```

The program will prompt the user to enter an `.xml` file:

```
Enter .xml file: datafile.xml
```

If the provided file is not found, or an incorrect file extension is given, an error message will display, and the user can re-enter a filename. If no extension is specified, the program will assume it is `.xml`. If the output files already exist in the directory, the program will overwrite the data currently on the files.

1.2 Phase 2: Building Indexes

Phase 2 takes the output files from phase 1 and sorts them, and produces four index files from the sorted data: `te.idx`, `em.idx`, `da.idx`, `re.idx`. If the index files already exist from a previous run, the data on them will be overwritten.

The index files are then loaded into four databases: `terms.db`, `email.db`, `date.db`, `recs.db`. Once again, if these databases are already found in the directory, they will be overwritten by the new data.

No user input is required for phase 2. To build the indexes, simply execute the following command in the terminal:

```
$ python3 phase2.py
```

1.3 Phase 3: Data Retrieval

Phase 3 provides the user a simple interface to query the data prepared in phases 1 and 2. The query program uses *Berkeley DB* to process the queries entered by the user. The user can query dates, emails, or terms. Depending on the type of query, it will have different formats. In general, queries are of the form:

```
> prefix:query
```

A date query will return all emails with dates that satisfy the query.

```
> date(:|>|<|>=|<=)YYYY/MM/DD
```

An email query returns all emails sent to/from/cc/bcc the email address specified:

```
> (to|from|cc|bcc):email@address.com
```

A term query returns all records that have the term in their subject or body field (field specified by user)

```
> (subj|body):term
```

Terms can also be queried without specifying the field. For example,

```
> confidential%
```

will return all records that have a term with prefix *confidential* in their subject or body fields (confidential, confidentially, confidentiality, etc.).

The program will ignore all white spaces in the entered queries, so

```
> subj :   gas          body:earning
```

will successfully search the data for records that have gas in their subject and earning in their body fields. All queries are case-insensitive as well.

2 Query Algorithms

The query algorithm used in this program operates by first evaluates each query the user enters individually,then takes the intersection of all queries to generate a master list of row IDs. Each ID in the master list is used to complete an equality search on the records database to generate the results that match all of the queries.

This algorithm is efficient in that it eliminates redundancy, while retaining accuracy. Since each query is completed at least once we can guarantee no results are missed, and since each query is completed at most once (including the query of the records database with the rowIDs), we are reducing the amount of redundant searches. One caveat of this system is that it does check to see if a key was already searched for (for example, if an equality search was already completed in a previous range search). This feature could be implemented at the cost of space.

A separate strategy is used to calculate each query depending on if the query requires an equality or range search of the items in the database, and if the query is a term query containing the wild card *(%)* operator or not.

- Equality Search

  The program executes equality searches by setting the cursor of the database to the provided key (term, date or email term), then adding all items to a set (to prevent redundancy) that have the same key (iterated by using the *cursor.next_dup()* command). The resulting set is returned.

- Range Search

  There are two types of range search: a greater-than search and a less-than search. Both utilize a similar strategy but the program distinguishes between both for simplicity:

  - Greater-Than Search

    The greater-than search first moves the cursor to the specified key (or the value directly after the key if it is not inclusive), then moves the database cursor along the rest of the database. Each item is added to a set of RowIDs as the database is traversed.
  - Less-Than Search

    The less-than search first moves the cursor to the first key in the database, then moves the database cursor along the database until the cursor reaches the key. Each item is added to a set of RowIDs as the database is traversed. If the search is exclusive, the query does not add the items corresponding to the provided key.

- Wild Card

  Wild card searches are handled similarly to range searches in that the database is traversed along a set of key-value pairs, with each value being added to a set. To do this the program moves the cursor to the specified key (without the wild card) and traverses the database until it reaches a key that does not have the provided key as a prefix.

3 Testing Strategy

All functions were tested extensively throughout development, and again after completion of the program. Both the brief and full output modes were tested, as well as those cases in which no results exist. Specific general test cases are listed below.

- Individual & Multiple Term Queries (combination of subj:term, body:term, term and term% on same line)
- Individual & Multiple Date Queries (combination of date:key, date<key, date<=key, date>key and date>=key on same line)
- Individual & Multiple Email Queries (combination of to:key, from:key, cc:key and bcc:key on same line)
- Individual & Multiple Combination of Queries (combination of different term, email and date queries on same line)

4 Group Work

- Phase 1: Dalton: 2.5 hrs
- Phase 2: Harrison & Dalton: Collaborative, 1 hrs
- Phase 3: Get Queries, Display Results (Brief and Full Output), Query Records, Get Field Functions, Parsing Queries
  - Dalton: 1 hrs
- Phase 3: Email, Term and Date Queries, Equality and Range Search Functions:
  - Harrison: 3 hrs
- Design Document
  - Dalton: 1.5 hrs
  - Harrison: 1 hrs
- Testing and Code Quality
  - Dalton: 0.5 hrs
  - Harrison: 1.5 hrs