CMPUT 291 Mini-Project 2 Report

Overview

The purpose of this project was to gain experience working with data in the physical layer. A data retrieval system is designed using the Berkeley DB library for operating on files and indices.

Mini-Project-2 is a python command-line application that interfaces with the Berkeley DB Python 3 package (bsddb3). Using the program users can specify queries written in a query language grammar. These queries are processed by the program and the associated data is retrieved and presented to the user.

User guide/Detailed design

Note: Our official user guide and documentation for mini-project-2 is located at: https://CMPUT291PROJECTF18-mini-project-2.readthedocs.io/en/latest

Main (Phase 3)

The main module of Mini-Project-2 is run as a command-line program. Information for using the program can be found by typing "mini-project-2 --help". The program is designed to output database information based on queries inputted to the main module. This is achieved by inputting "mini-project-2 <flags> <query(s)>". The query language grammar supported by Mini-Project-2 is specified by the following BNF:

```
alphanumeric    ::= [0-9a-zA-Z_-]
numeric         ::= [0-9]
date            ::= numeric numeric numeric numeric '/' numeric numeric '/' numeric numeric
datePrefix      ::= 'date' whitespace* ('=' | '>' | '<' | '>=' | '<=')
dateQuery       ::= datePrefix whitespace* date
price           ::= numeric+
pricePrefix     ::= 'price' whitespace* ('=' | '>' | '<' | '>=' | '<=')
priceQuery      ::= pricePrefix whitespace* price
location        ::= alphanumeric+
locationPrefix  ::= 'location' whitespace* '='
locationQuery   ::= locationPrefix whitespace* location
cat             ::= alphanumeric+
catPrefix       ::= 'cat' whitespace* '='
catQuery        ::= catPrefix whitespace* cat
term            ::= alphanumeric+
termSuffix      ::= '%'
termQuery       ::= term | term termSuffix
expression      ::= dateQuery | priceQuery | locationQuery | catQuery | termQuery
query           ::= expression (whitespace expression)*
```

The main module will process the request, determine an efficient way to retrieve the desired data, and then output the retrieved data to the user.

The flags that can be inputted to the main program include:

- -o (or --output) which can be set to "full" or "brief". This flag determines the verbosity of the outputted data [OPTIONAL, default= "brief"].
- -ad (or --ads-index) which specifies the path to the ad.idx file [REQUIRED].
- -te (or --terms-index) which specifies the path to the te.idx file [REQUIRED].
- -da (or --pdates-index) specifies the path to the da.idx file [REQUIRED].

- -pr (or --prices-index) specifies the path to the pr.idx file [REQUIRED].

Parser (Phases 1 and 2)

The parser module uses a combination of built-in Python XML parsing and line-by-line RegEx parsing to parse inputted XML. Based on the XML records, text files are created for creating query indices in phase 2. To run the parser module, navigate to the directory of parser.py and run "python parser.py". A prompt will ask for the path to the XML data file. After this is inputted, the parser will output 4 text files. A shell script can then be used to generate 4 Berkeley Database index (.idx) files used in querying data in the main module. The .idx files can be created by running our shell script "**gen_idx.sh**" within the mini-project-2/scripts/ directory.

Testing strategy

Our primary testing strategy was to employ white-box unit testing using the pytest module. Our unit testing would validate the operation of all functions (internal and external) by comparing them to expected outputs.

Our testing efforts can be found in the https://github.com/CMPUT291PROJECTF18/Mini-Project-2/tree/master/test/unit subdirectory of our code repository.

To keep our master branch stable, a continuous integration service was employed. Our CI was run for every pull request to ensure that our code maintained its quality and stability. Past and present testing results can be seen on our Travis CI build history at:

https://travis-ci.com/CMPUT291PROJECTF18/Mini-Project-2

Breakdown of work

Nathan Klaptstein (nklapste)

- Hours: ~16 hours
- Progress: Provided the boilerplate for the project. Developed phase 2. Co-developed phase 3. Contributed unit tests for each phase.

Thomas Lorincz (tlorincz)

- Hours: ~16 hours
- Developed functionality for project phase 1. Co-developed phase 3. Contributed unit tests for each phase. Wrote the project report.

Work coordination

We primarily coordinated through GitHub. An organization was set up for the project (CMPUT291PROJECTF18) and each task or bug was turned into an issue ticket. These tickets were assigned to each other and each pull request was associated with an issue. To measure progress, and when an issue was done, the assigned member would close the issue. This workflow increased productivity immensely and did not result in codebase errors (all users were responsible with their git practices).

Communication was performed regularly over Slack. We met at least once a week to work on the project for a few hours.

Design decisions differing from requirements

None that we are aware of.