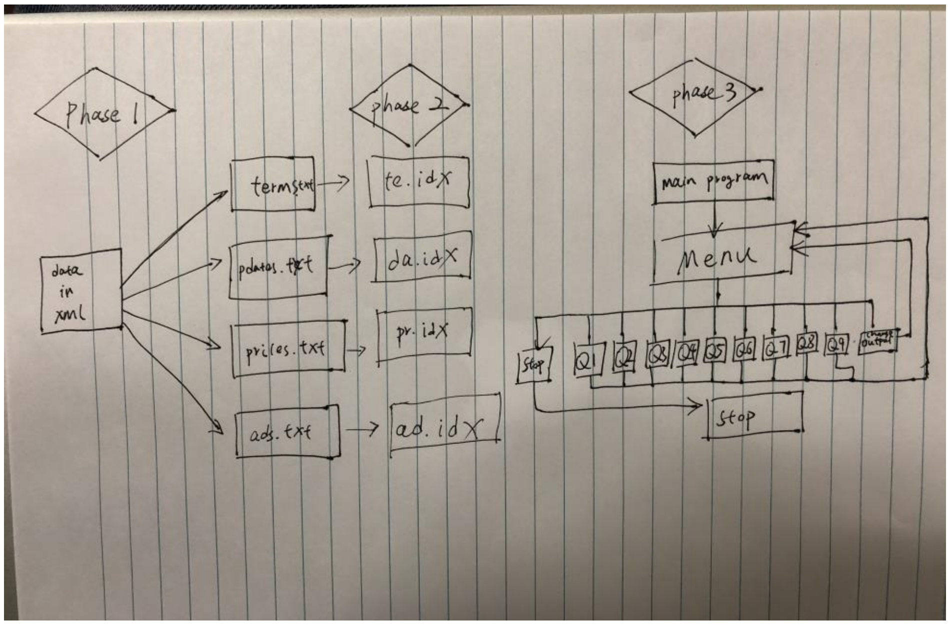
Design Document

General overview of the system / User Guide

The application contains three phases. The first phase will take an xml file as input and it will generate four .txt files. The second phase will sort the files that generated in phase 1 and it will also create 4 index files. The third phase will allow users to do some queries. The application is mostly programmed in python, and uses its bsddb3 library in order to store data in Berkeley DB. A Linux shell script is also included, for parsing the data into .idx files.



Software Design Components

PHASE 1:

Main Function:
The main function takes an xml file as input and generate four txt files *(terms.txt,pdates.txt, prices.txt, ads.txt*). the Main function uses the following sub-classes for its functionality:

Get_content:
This function will parse the xml file. This function will take three arguments as input: File name, instantiation (<>) tag, and the ending (</>) tag. The content in the tag will be returned.

Write_terms:
This function will take the xml file name as input. It will generate a file called terms.txt. The content of terms.txt is the content from tag <ti> and <desc>. All the malformed characters and the words shorter than 3 characters will be removed.

Write_pdates:
This function takes the xml file name as input. It generates a file called pdates.txt. The content of pdates.txt is that from the <aid>, <date>, <cat>, and <loc> tags.

Write_prices:
This function will take the xml file name as input. It will generate a file called prices.txt. The content of prices.txt is the content from tag <aid> <price><cat><loc>.

Write_ads:
This function will take the xml file name as input. It will generate a file called ads.txt. The content of ads.txt contains one line for each ad.

PHASE 2:

sort.sh:
Batch shell script responsible for creating Berkeley DB files (formatted as .idx) from their respective input text files. The script has a dependency on the Perl script break.pl, which is used for formatting the txt files, and preventing malformed input due to invalid characters. The script may be called from a bash terminal. The shell script uses the sort, uniq, and db_load functions to generate 4 .idx indexes. ad.idx in created in hash and other indexes are created in b_tree.

PHASE3:

Phase 3 will consist of a python script, which will be responsible for taking user input, parsing it into a query expression, and outputting the result of said query when run against the provided database.

Tokenizer:
This function is used to parse the raw user input into tokens readable by the actual parser. The function will use regular expressions to format the tokens into their proper category. As defined in the grammar, the script will recognize alphanumeric, and numeric characters; keywords, which define the keys used in an expression; and operators, which define comparisons with the characters.

Parser:
Using the tokens generated from the above function, this will be responsible for recursively performing expressions which will generate an output. Unary, and Binary operations occur within the queries, and as such this function must handle their formats correctly, and pipeline them to other portions of the expression. Once the unionized query is properly concatenated using appropriate set operations, the function will return a python list containing all the matches.

Printer:
This function will present the output of the parser in a human-readable format, in a table with header labels, and the ability to scroll through the results in a comprehensive method.

Testing Strategy

The testing strategy of our group will consist of a series of queries, in which we will analyze the system's functionality, and test for edge cases which can produce undocumented behaviour. The example queries outlined within the project specification will be used to develop the system, and then test whether the outputs match what is expected.

Malformed queries will also be used within the input, such to test whether the system is able to recognize and terminate any queries which may produce undocumented behaviour.

If exhaustive testing is warranted, and time is not an asset, cross-referencing the queries with an SQL database could be used to ensure that the queries are accurate. As such, this would require a script to convert the database to an SQL-readable format.

Group Work Breakdown Strategy

Group coordination was established using GitHub, to track progress, and ensure that overlapping code can be dealt with effectively, such to ensure the most effective solution is recorded. GitHub allowed group members to simultaneously work on separate parts of the code, and commit their work to a central, working version.

Group time allocation:

Linhao Xian: 20 Hours

Seiver Bredeson: 17 Hours

Work Breakdown:

Linhao was responsible for creating the phase 1 XML parser, which entailed opening the database, parsing the content using regular expressions, and sorting the content of the databases into the respective text files. Seiver created the phase 2 shell script which formats the text files into Berkeley Database files, and provided feedback and debugging for issues and bugs which came up throughout the project. Both Linhao and Seiver have progress made into phase3 of the project, of which is included in the submission.