CMPUT291 Project2 Document

Group Members: Zhijie Shen, Liyao Jiang, Hongru Qi

System Overview:
The program takes a given XML format data file, and parse it and builds indices. And then takes input queries and process them.

The phase2.sh script runs the whole program, it will call the phase1.py to parse the given XML file, and generates the .txt files. Then it uniquely sorts the .txt files and pipeline them in stdout to the given Perl script to break them into the db_load expected format, and use db_load to build the specified type (hash or btree) of indices.idx files. Finally, it calls the getQueryInput.py to ask for input and evaluates queries, and the program will print out the results.

User Guide:
To test all three phases of this system, go to command line and run shell script by the following command: **'sh -e phase2.sh xxx.txt'**
Where the (xxx.txt) would be replaced with the name of the desired XML file.
For example: **'sh -e phase2.sh 10.txt'**

Test phase1 only, use '**python3 phase1.py xxx.txt'** (parses the XML, generates the .txt files)
Test phase3 only, use '**python3 getQueryInput.py'** (asks input then evaluates queries, need to have the .idx files ready in the outputs folder)
In phase 3, type **'output=full'** or **'output=brief'** to choose the output format, full will be chosen by default.

Software Design:
Range searches and exact searches on the different columns are handled by separate functions, each condition is parsed by the main program into tokens, each function uses Berkley DB cursor to iterate through the .idx files, and compares the result with the input, then returns all the qualified entries.

For a multi-conditional query, the searches on key columns (i.e. term, partial, term, price, date) are performed first then we take the set intersection of all the results returned by these functions. Then the program performs searches on cat and location condition, and the search is within the previous result set to increase efficiency. Also, the search range is reduced after the evaluation of each condition by passing the result range to the next evaluation. When there are no key column conditions, the search is started with the range of the whole database.

For the wildcard partial match, we use the Berkley DB cursor.setrange to do a range search and use a while loop to find all the records and break the loop until the next record's substring[0:len("partial term")] value > the given partial term string value.

Testing strategy:
Because each function is implemented separately, testing is also done separately. Each functionality is tested through making test functions inside main() for each .py file to make sure the functions are behaving as intended. For the query conditions evaluation function of different types (e.g. term, partial term, date, price), we test each function on the specific type and then test the functions on multi-conditional queries in the getQueryInput.py file.

Group work break-down:
The project has 3 phases, we discuss together to find feasible solutions for each phase and then split the work, and each feature is implemented separately and combined together at the end. Here is the project break-down between partners.

Each member implemented and tested the followings:
Zhijie Shen: phase1.py, collaborated on getQueryInput.py (getting the input and breaking down keywords
Liyao Jiang: phase2.sh(shell script, sort and break the .txt and generate the .idx files), queryEval.py(handles date and price conditions of the queries), collaborated on getQueryInput.py (worked on the result sets intersection)
Hongru Qi: nonkeySearch.py, collaborated on getQueryInput.py (re-organize input keywords into date, price, term, partial term, and nonkey conditions as well as call corresponded functions to find all matched aids in the most efficient order)

Time allocated and spent for each feature and progress made: we are expecting about 3 hours each for the three phases. We assigned subtasks to each member at the beginning and started to implement each feature, each member is also responsible for testing the functions. When we each completed the implementation and testing of the assigned feature, then we assigned the rest subtasks to each member. When challenges are encountered, we pair up and try to solve the challenge. After finished all the subtasks, we work together to combine all the query functions in the getQueryInput.py file, and modified the phase2.sh script to run the whole program by calling the all the components.