Project Report

Logan Yue
Thomas Maurer
Jolene Poulin

General Overview:

Section 1: Phase 1, run with the command “python3 phase1.py [RECORD_FILE]”, loads the argument xml file and parses it line by line. It splits each line into its components and then saves the correct components to the 4 required text files (ads.txt, pdates.txt, terms.txt, and prices.txt).

Section 2: Phase 2, run with the command “python3 phase2.py” takes the files created by phase 1 and converts them into .idx files. It does this by sorting the files using linux commands and then splitting the sorted files into key value pairs as expected by the db_load commands. It then calls the db_load command with these ready files.

Section 3: After creating the idx files, the user can execute queries by running: “python3 queryDB.py”. This interface will provide the user with the ability to query the system using any of the queries specified within the provided formatting. Once the user is done, they may input “exit” to close the program.

Design:

Setting up the database: First the input file will be read and parsed into organized text files. This is done through simple regex checking on each line in the file. In phase 2 the idx databases are initialized with the proper flags. Then the files from phase 1 are sorted with the linux sort command and duplicate rows are removed. This creates sortedAds.txt etc. These sorted files are then formatted through more regex to create ready files which contain key value pairs on separate lines. These are then passed to the db_load linux command to populate the databases created earlier.

Querying the database: Each search type is contained within its own function and is called on by the interface individually. There is also one extra search function that is given a list of object ids and returns a list of all of the values related to each object.

get_full_data(): When querying ad.idx to get full data, we use database.get(id) to retrieve the values.

search_date() and search_price(): Queries da.idx and compares each date to the provided date and acts according to the given operator

search_term(): Queries te.idx using cursor.set-range() and sets cursor as the first match in the index and adds them to the data until a mismatch is found and then stops.

Search_cat and search_location(): Because these values are not keys in any index, the functions iterate over ad.idx and match the given value. This decreases runtime by removing the necessity to later query the ad.idx using get_full_data()

Multiple Query Conditions: For all queries, the query is separated into words by spaces or operators (><=). These are then evaluated one at a time. If a condition requires an operator and a target, then the words will be combined and sent to a search function. The search functions are passed the condition and any data from a previous search. If there is data present from a previous search, in order to reduce runtime, the search checks the condition against the elements in the given data instead of checking the database reducing the number of elements to be checked as equal to the number of elements from the previous set of data. This is constant and so runtime will increase less than linearly with the number of conditions as each condition will reduce the number of objects to iterate over. Worst case however is O(c * n) where n is the number of elements in the database and c is the number of conditions.

Testing Strategy: The majority of our testing was done through guess and check. To test major functionality we ran each portion on the correct files from 1000 records. Then we tested with 100k record files to test time consumption. The files were compared against the correct files with diff and db_dump.

Workload Distribution:

Logan Yue:
Worked on: Multiple querying conditions as well as reformatting single query conditions and writing search functions for locations and categories. Also debugged date search and term search. Provided the final interface which involves parsing user input and calling search functions accordingly. Also wrote all search functions for when data is provided instead of searching the database.
Approximate time spent: 11 hours
Progress made: Completed functionality of all of the functions mentioned in worked on

Jolene Poulin:
Worked on: Single term search and ground work for interface. That is, queries against only one database and how to decide which database to open. Full versus brief output functionality. Finding search terms and comparison values in an input string.
Approximate time spent: 4 hours
Progress made: Everything mentioned above

Tom Maurer:
Worked on: Phase 1 and Phase 2
Approximate time spent: 10-12 hours
Progress made: Completed phase 1 and phase 2 as well as some debugging and optimizations on search functions