Overview:

This project works with data in the physical layer by building an information retrieval system using the Berkeley DB library for operating on files and indices. Within the project there will be a program that keeps data in files and maintain indices which can be used to provide basic searches over such data.

Most inputs will follow the general format:

expression	::= dateQuery | priceQuery | scoreQuery | termQuery
query	::= expression (whitespace expression)?
modeChange	::= 'output=full' | 'output=brief'
command	::= query | modeChange
dateQuery	::= ‘date’ + ('>' | '<')+ YYYY/MM/DD
priceQuery	::= (price) + ('>' | '<') + numeric
scoreQuery	::= (score) + ('>' | '<') + numeric
termPrefix	::= (pterm | rterm) + ':'
termSuffix	::= '%'
termQuery	::= termPrefix? + term termSuffix?

‘+’ refers to having zero or more white spaces
‘?’ refers to being optional

Within the expression, there maybe one or more of the following: a date query, price query, score query, term query. Each individual query is separated by one or more white spaces. The term query is restricted to searching for only alphanumeric terms. The score/price take only numeric values and date requires the ‘YYYY/MM/DD’ format to operate. Users may also choose to switch between ‘full’ or ‘brief’ mode to toggle the output format of each future query. To exit the program enter ‘exit()’.

Software design:

The algorithm for efficiently evaluating queries involves breaking down the inputted command into separate functions to parse the data individually, eventually intersecting the results from each separate function to obtain the expected outcome. Queries with multiple conditions were handled by parsing the initial inputted command using a python library called ‘re’ (regular expression) to split the one-line command into individual queries. The wild card ‘%’ was handled with a simple conditional statement when searching for a term which will turn the condition true if the term to search for ends with ‘%’. Range searches were handled by creating an upper and lower bound. Using the python library ‘math’, we are able to set an upper bound of infinity to maintain comparisons that did not include an upper bound. Overall, our program efficiently parses the user input and is robust enough to handle all valid/invalid user input.
Assumptions: all numerical values that are to be inputted are restricted to being positive.

Testing Strategy:

All testing was done using the test data in the reviews.txt, scores.txt, pterms.txt, rterms.txt files which were processed into separate. idx files to be used in our program. The queries were tested individually as the program was being written. Once the functions where completed, more testing was done using valid/invalid combinations of queries. The correctness was judged by manually examining the results from the program and comparing with the test data to ensure the code's correctness.

Break-down:

ducay - Succesfully managed to complete the rterm/pterm queries. woldegio - Succesfully managed to complete the score and price queries. Ojohnson - Succesfully managed to complete phase 1 and the time queries. The group members shared files through Github in order to ensure efficient collaboration as well as to efficiently distribute the work load. The group met up two times for an average of one hour in-call to discuss the project. Each group member spent approximately 9 hours over the course of a week completing their work load.