## **Mini-project 2 spec**

# CMPUT291 - Fall 2018 Mini Project II

## (group project)

***Due****: Nov 26th at 5pm*

### **Clarifications**:

You are responsible for monitoring the course discussion forum in eclass and this section of the project specification for more details or clarifications. No clarification will be posted after *5pm on Nov 25th*.

* **Nov 24 (the .idx files)** They are added to the directories on the lab machines.
* **Nov 22. (a fix in dataset)**In our 20k dataset, some of the dates were not in the right format. They should be fixed now.
* **Nov 21.** (**a clarification in Phase 1*)*** When constructing our terms.txt, we ignore **&#numbers;**  (i.e. replacing them with empty strings) while we replace **&apos;**, **&quot**; and **&amp;**withrespectively ', ", and &, which are later treated as separators.
* **Nov 21**. (**a change in dataset)** I heard from a few groups that they had difficulty indexing price. The problem is that Berkeley DB orders the prices lexicographically and this does not give the right ordering when searching based on price (e.g. 1 is listed after 1000). To avoid this problem, I am changing prices.txt, padding spaces at the beginning. This should hopefully help.

### **Introduction**

The goal of this project is to teach the concept of working with data in the physical layer. This is done by building an information retrieval system, using the Berkeley DB library for operating on files and indices. Your job in this project is to write programs that keep data in files and maintain indices that provide basic searches over data. 80% of the project mark would be assigned to your implementation which would be assessed in a demo session, and is further broken down to three phases with 10% of the mark allocated for Phase 1, 5% of the mark for Phase 2, and 65% for Phase 3. In addition to the correctness of your indices and your query results, the efficiency of your programs and your queries (in terms of the running time) will be evaluated. Another 15% of the mark will be assigned for the documentation and quality of your source code and for your design document. 5% of the mark is assigned for your project task break-down and your group coordination.

### **Group work policy**

You will be doing this project with one or two other partner from the 291 class. Register your group at the [group registration page for mini-project 2](https://eclass.srv.ualberta.ca/mod/data/view.php?id=3310391). It is assumed that all group members contribute somewhat equally to the project, hence they would receive the same mark. In case of difficulties within a group and when a partner is not lifting his/her weight, make sure to document all your contributions. If there is a break-up, each group member will get credit only for his/her portion of the work completed (losing the mark for any work either not completed or completed by the partner).

### **Task**

You are given a data file, which you will use to construct your indices. Here is a small data file with only [10 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/10.txt) and here is one with [1000 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/1k.txt). The data includes ads posted on Kijiji; each record consists of an id, a category, a location, a title, a description, a price (not all ads may have a price or the price may be 0) and a date when the ad is posted. The records are formatted in xml and have a start tag <ad> and and an end tag </ad>; each record has a newline character at the end, which can help you easily process the file (without using an xml parser). Fields inside records are also formatted similarly with respective tags. Your job is to create indices, following Phases 1 and 2, and use those indices to process the queries in Phases 3.

### **Phase 1: Preparing Data Files**

Write a program that reads records in xml from the standard input and produces 4 files as described next. The format of the input is as given in the samples with the first, second and the last input lines giving the xml tags and every other line giving a record. Because of this simplicity, you don't need an xml parser for the input. Here are the files you will be generating in Phase 1.

**terms.txt**: This file includes terms extracted from ad titles and descriptions; for our purpose, suppose a term is a consecutive sequence of alphanumeric, underscore '\_' and dashed '-' characters, i.e *[0-9a-zA-Z\_-]*. The format of the file is as follows: for every term*T*in the title or the description of an ad with id *a*, there is a row in this file of the form *t:a*where *t*is the lowercase form of *T*. Ignore all special characters coded as *&#number;*such as &#29987; which represents *産*as well as &apos;, &quot; and &amp; which respectively represent ', " and &. Also ignore terms of length 2 or less. Convert the terms to all lowercase before writing them out. Here are the respective files for our input files with [10 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/10-terms.txt) and [1000 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/1k-terms.txt).

**pdates.txt**: This file includes one line for each ad in the form of *d:a,c,l*where d is a non-empty date at which the ad is posted and *a*, *c*, and *l*are respectively the ad id, category and location of the ad. Here are the respective files for our input files with [10 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/10-pdates.txt) and [1000 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/1k-pdates.txt).

**prices.txt**: This file includes one line for each ad that has a non-empty price field in the form of *p:a,c,l*where p is a number indicating the price and *a*, *c*, and *l*are respectively the ad id, category and location of the ad. Here are the respective files for our input files with [10 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/10-prices.txt) and [1000 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/1k-prices.txt).

**ads.txt**: This file includes one line for each ad in the form of *a:rec*where *a*is the ad id and rec is the full ad record in xml. Here are the respective files for our input files with [10 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/10-ads.txt) and [1000 records](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/1k-ads.txt).

These files can also be found at directory ~drafiei/291/pub on the lab machines. In the same directory, you would also find larger size files (with 20k records or larger) that you may want to use in the testing of your programs.

### **Phase 2: Building Indexes**

Sort the files built in Phase 1 using the Linux *sort*command; pass the right option to the sort command to keep only the unique rows (see the man page for sort). You can keep the sorted data under the same file names or pass the sorted records to stdout so they can be piped to your loading program (as described next). Suppose the sorted files are named as before (to simplify our presentation here). Given the sorted files *terms.txt*, *pdates.txt*, *prices.txt*and *ads.txt*, create the following four indexes: (1) a hash index on ads.txt with ad id as key and the full ad record as data, (2) a B+-tree index on terms.txt with term as key and ad id as data, (3) a B+-tree index on pdates.txt with date as key and ad id, category and location as data, (4) a B+-tree index on prices.txt with price as key and ad id, category and location as data. You should note that the keys in all 4 cases are the character strings before colon ':' and the data is everything that comes after the colon. Use the *db\_load*command to build your indexes. db\_load by default expects keys in one line and data in the next line. Also db\_load treats backslash as a special character and you want to avoid backslash in your input. [Here](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/break.pl) is a simple Perl script that converts input records into what db\_load expects and also removes backslashes. Your program for Phase 2 would produce four indexes which should be named *ad.idx*, *te.idx*, *da.idx*,  and *pr.idx*respectively corresponding to indexes 1, 2, 3, and 4, as discussed above. It can be noted that the keys in *ad.idx*are unique but the keys in all indexes can have duplicates.

In addition to *db\_load*, you may also find *db\_dump*with option *p* useful as you are building and testing the correctness of your indexes.

### **Phase 3: Data Retrieval**

Given the index files *ad.idx*, *te.idx*, *da.idx*and *pr.idx*created in Phase 2 respectively on ad ids, terms, dates, and prices, write a program that efficiently processes queries as follows. By default, the output of each query is the ad id and the title of all matching ads. The user should be able to change the output format to full record by typing "*output=full"* and back to id and title only using "*output=brief"*. Here are some examples of queries:

1. *camera*
2. *camera%*
3. *date <= 2018/11/05*
4. *date > 2018/11/05*
5. *price < 20*
6. *price >= 20*
7. *location=edmonton date=2018/11/07*
8. *cat=art-collectibles camera*
9. *camera date>=2018/11/05 date<=2018/11/07 price > 20 price < 40*

The first query returns all records that have *camera*as a word in the title or description fields. The second query returns all records that have a term starting with *camera*in the title or description fields. The third and the fourth queries respectively return ad records that are posted on or before 2018/11/05, and after 2018/11/05. The fifth and the sixth queries respectively return ad records with prices less than 20, and greater than or equal to 20. The seventh query returns all ads from Edmonton posted on 2018/11/07. The eight query returns all ads in category art-collectibles that have the term camera in the title or description fields. Finally the ninth query returns all records that have the term camera in the title or description, are posted on or after 2018/11/05 and on or before 2018/11/07, and have a price greater than 20 and less than 40.

More formally, each query defines some conditions that must be satisfied by title, description, post date, location, category and the price fields of the matching records. A condition on terms can be either an exact match (as in Query 1) or a partial match (as in Query 2); for simplicity, partial matches are restricted to prefix matches only (i.e. the wild card % can only appear at the end of a term). All matches are case-insensitive, hence the queries "Camera", "camera", "cAmera" would retrieve the same results; for the same reason the extracted terms in previous phases are all stored in lowercase. Conditions on dates and prices are both equality and range matches, specified using *<*, *<=*, *>*, *>=*, and *=*. Conditions on location and category are exact (but case-insensitive) equality matches. There is zero or more spaces between column names (e.g. *price*, *date*, *location*, and *cat*), and the comparison symbols (e.g. >, =) and the numbers, dates, or strings that follow; hence, *price<20*, *price< 20*, *price <20*, and *price     <    20*are all valid and would return the same matches. There is no space between < and = in <= and between > and = in >=. Each query term, location, and category name is a consecutive sequence of alphanumeric, underscore '\_' and dashed '-' characters, i.e *[0-9a-zA-Z\_-]*. A query term can also have the wild card symbol % at the end. A query can have multiple conditions (as in Query 9) in which case the result must match all those conditions (i.e. the *and*semantics), and there is one or more spaces between the conditions. The keywords *price*, *cat*, *location, date* and *output* are reserved for searches and query interface (as described above) and would not be used for term searches. The dates are formatted as yyyy/mm/dd in both queries and in the data. Here is the [query language grammar](https://eclass.srv.ualberta.ca/pluginfile.php/4443984/mod_page/content/46/ql-grammar.txt).

When a query can be evaluated using multiple strategies, your evaluation should select the most efficient strategy (i.e., the one expected to be the fastest) as much as possible. For example, for Query 1, you can use the index on terms to find the matching aids, then use the index on aids to find the actual records. For Query 7, you can use the index on date. Since the index on dates also stores locations, you can verify the location and only retrieve aids that match both the date and the location, before going to the index on aids to find the actual records.

### **Testing**

At demo time, your code will be tested under a TA account. You will be given the name of a data file, and will be asked (1) to prepare terms.txt, pdates.txt, prices.txt, ads.txt (2) build Berkeley DB indexes ad.idx, te.idx, da.idx and pr.idx, and (3) provide a query interface, which will allow us to test your system for Phase 3. We typically follow a 5 minutes rule, meaning that you are expected to prepare the data and build your indices in Phases 1 and 2 in less than 5 minutes; if not, you may lose marks and we may have to use our own indexes, in which case you would lose the whole mark for Phases 1 and 2.

The demo will be run using the source code submitted and nothing else. Make sure your submission includes every file that is needed. There will be a limited time for each demo. Every group will book a time slot convenient to all group members to demo their projects. At demo time, all group members must be present. The TA will be asking you to perform various tasks and show how your application is handling each task. A mark will be assigned to your demo on the spot after the testing.

**Important note**: You can make no assumption on the size of the input file (and any of the files and indexes that are built from the input file). Avoid using xml processors that may attempt to load the whole data file into memory. Each record is in one row and should be easy to process without such tools. We will be using a relatively large file for testing and you don't want your program to break down in our tests, resulting in a poor mark! That said, you can make very few assumptions (such as the inverted list of a term can fit in main memory) and state them in your report.

### **Instructions for Submissions**

Your submission includes (1) the source code for phases 1, 2 and 3 and any makefile or script that you may need to compile your code, (2) README.txt, and (3) a short report. Your source code must include at least three programs, i.e. one for each phase. Your program for Phase 3 would implement a simple query interface in your favorite programming language (e.g. Python, Java, C or C++). For phases 1 and 2, you can have more than one program (for example, in Python, Java, C, C++ or Perl) and can make use of any Unix command and scripting language (e.g. Perl, bash) that runs under Linux on lab machines, as long as you clearly document in your report how and in what sequence the programs or commands should run.

* Create a single gzipped tar file with (1) all your source code, (2) README.txt, and (3) your project report. Name the file *prj2.tgz*.
* Submit your project tarfile in the project [submission site](https://eclass.srv.ualberta.ca/mod/assign/view.php?id=3105843) by the due date at the top of this page.
* All partners in a group must submit their own copies(even though the copies may be identical)!

The file README.txt is a text file that lists the names and ccids of all group members. This file must also include the names of anyone you collaborated with (as much as it is allowed within the course policy) or a line saying that you did not collaborate with anyone else. This is also the place to acknowledge the use of any source of information besides the course textbook and/or class notes.

Your report must be type-written, saved as PDF and be included in your submission. Your report cannot exceed 3 pages.

The report should include (a) a general overview of your system with a small user guide, (b) a description of your algorithm for efficiently evaluating queries, in particular evaluating queries with multiple conditions and wild cards and range searches and an analysis of the efficiency of your algorithm, (c) your testing strategy, and (d) your group work break-down strategy. The general overview of the system gives a high level introduction and may include a diagram showing the flow of data between different components; this can be useful for both users and developers of your application. The user guide should have instructions for running your code for phases 1, 2 and 3. The testing strategy discusses your general strategy for testing, with the scenarios being tested and the coverage of your test cases. The group work strategy must list the break-down of the work items among partners, both the time spent (an estimate) and the progress made by each partner, and your method of coordination to keep the project on track. Your report should also include any assumption you have made or any possible limitations your code may have.