Programming Project #2 (Files and Indexing)

Programming Project #2 (Files and Indexing)

Attached Files: PHARMA_TRIALS_1000B.csv (74.141 KB)

Overview

The goal of this project is to implement a program that parses an ASCII text-based CSV data file and then transforms that same information to a memory-efficient *binary* format and writes it to a **binary data file**. The new binary data file shall also have multiple index files for efficient record retrieval of **values** from fields other than the primary key. Your program should operate entirely from the command line (no GUI).

Requirements

- It is highly suggested to use **Java** for this project, but you may also use **C**, **C++**, or **Python**.
- Your program should be named **MyDatabase**. If you use Java, the main method should therefore be contained in a file named **`MyDatabase.java`** (assuming you use Java).
- Your program should launch and then present the user with an text-based menu that provides the following options. The possible commands fall into four categories:
  - Import
  - Query
  - Insert
  - Delete

Import

- The import command should parse and import an input CSV file to a binary data file. The CSV file will be provided. Your program *must* store its binary database in a single file named "*`table_name`***`.db`**". Your binary data file must conform to a pre-defined schema. The data file *cannot* use record separator characters (e.g. newline, line feed, etc.), or field separators (e.g. commas, vertical bars (|), etc). Record location within the file must be stored in index files. Field location within record are known for fixed size data types and stored in a size byte for varchar.
- The import command should have the syntax:
  - **`prompt> import`** *`file_name`***`.csv`**

Query

- The syntax for queries should emulate SQL, i.e.
  - **`SELECT`** `[comma_separated_field_list | *]`
    `FROM` `PHARMA_TRIALS_1000B`

WHERE *field_name* [**NOT**][=|>|≥|<|≤] *value;*

- You are only required to support a single *table* query. ("PHARMA_TRIALS_1000B" in this example).
- The table name should be the same name as the CSV from which it was imported, i.e. importing the CSV file named "PHARMA_TRIALS_1000B.CSV" should result in a binary data file named "PHARMA_TRIALS_1000B.db", which represents a table named "PHARMA_TRIALS_1000B".
- You are only required to support a single *field_name* in the query **WHERE** clause.

- Create one index file for each field. These should be created at the same time as the binary data file. Index files should be named for the table and field(s) being indexed and have the extension .ndx, e.g. **PHARMA_TRIALS_1000B.id.ndx**, **PHARMA_TRIALS_1000B.company.ndx**.
  - Each index file should have one entry (one line) for each *unique* value in that field. Therefore, the **id** index (primary key) will have 1000 entries. Other fields may have fewer entires. The boolean fields will only have two entries (one for true, one for false).
  - For each entry in a given index file, a list of **binary locations** (i.e. offset number of bytes from the file beginning) for all records that contain that index value will follow.
  - Index file format may be either plain text (ASCII) or serialized binary objects, your choice. If you use plain text files for indexing, you *may* use newlines in the index files to separate each entry.

Insert

- .The syntax for insert should emulate SQL, i.e.
  - **INSERT INTO** table_name **VALUES (***values_list***);**
- A new record created by the INSERT command should be appended on to the end of the data file. However, each new record will require new entries in the appropriate sort order place of all associated index files.

Delete

- The syntax for insert should emulate SQL, i.e.
  - **DELETE FROM** *table_name* **WHERE** *field_name* [**NOT**][=|>|≥|<|≤] *value;*
- Like queries, it needs only to support a single field_name in the query WHERE clause.
- Delete should be implemented by setting a single bit from 1 to 0 in the Boolean byte, i.e. record rows should not be physically deleted from the .db binary data file, but a 0 delete bit value should filter it from queries.

Data Schema Info

A CSV data file is attached: PHARMA_TRIALS_1000B.csv

Your program should transform the data into binary file where fields are converted to the binary types indicated below.

- The **varchar** field in the binary data file should be prefixed with a single byte that indicates the length of the string.
  - e.g. the string "GlaxoSmithKline LLC" should be converted to binary with the byte value **0x13** (i.e. "19", the length of the string) prefixed as a single byte.
  - Hexadecimal: 0x13 0x47 0x6c 0x61 0x78 0x6f 0x53 0x6d 0x69 0x74 0x68 0x4b 0x6c 0x69 0x6e 0x65 0x20 0x4c 0x4c 0x43
  - Chars: "<DC3>GlaxoSmithKline LLC", where <DC3> is the non-printable ASCII char 'device control 3'.

- Note that the four **Boolean** fields should all be stored in the same byte. The first four most significant bits should always be false (i.e. 0000, hexadecimal 0x0). The next four bits should store the boolean values of the four fields respectively, in order. For example: double_blind=**true**, controlled_study=**true**, govt_funded=**false**, and fda_approved=**true** would have the bits 00001101, which should be stored as the single byte **0x0d**.
- Therefore, record id=992 should take up a total **41 bytes** = 4 + (1 + 19) + 6 + 2 + 2 + 2 + 4 + 1 in the binary file.
    - It uses **70 bytes** (including the newline) in the CSV file).

Field	Data Type	Binary Size
id	integer	4 bytes
company	varchar	variable (1 length byte + 1 byte/char)
drug_id	char(6)	6 bytes
trials	short int	2 bytes
patients	short int	2 bytes
dosage_mg	short int	2 bytes
reading	float	4 bytes
double_blind	Boolean	1 byte
controlled_study	Boolean	
govt_funded	Boolean	
fda_approved	Boolean	
unused	—	
unused	—	
unused	—	
deleted	Boolean	

Submission

- Submit your source code file(s), your binary data file, and your 11 index files as a zip'd or tar'd file with your NetID as a name, e.g. cid021000.zip, cid021000.gzip, cid021000.tar.
  Include a readme file that describes how to compile your code. Include language, version, and compile command string:
    - e.g. prompt> **javac MyDatabase.java**
- You do not need to submit the input CSV data file.

Binary Data File Example

Assuming that you are importing the CSV records in the same order as given, the first four records should have the following Hexadecimal byte values when viewed in a hex editor. I have color coded the records so that you can see where the delineation between records is. Note that bytes are in groups of four, in a similar format to most Hex editors. Actual binary files have no such four byte grouping or spaces.

```
00000381 13446173 616e2045 26542043 6f2e2c20 4c74642e 4c452d31
31310010 07ED01DF 42BE6666 0A000003 94154d61 6a6f7220 50686172
6d616365 75746963 616c7345 4d2d3535 34005705 0B01B141 30000009
00000027 1C506861 726d6163 69612061 6e642055 706a6f68 6e20436f
6d70616e 7956422d 35313700 4404E001 2E427266 66030000 03311643
```

```
7573746f 6d204842 4320436f 72706f72 6174696f 6e4f4e2d 39383900
3705BA00 6E419F33 3306
```

Here is the data in more **human readable form**. with the original CSV text separated by tabs (but with double quotes stripped). Below the CSV text is the binary equivalent whose fields are separated by tab characters.

Record id=897 is at byte location 0
897,Dasan E&T Co., Ltd.,LE-111,16,2029,479,95.2,true,false,true,false
00000381 13 446173616e2045265420436f2e2c204c74642e 4c452d313131 0010 07ED 01DF 42BE6666 0A

Record id=916 is at byte location 41
916,Major Pharmaceuticals,EM-554,87,1291,433,11.0,true,false,false,true
00000394 15 4d616a6f7220506861726d6163657574696363616c73 454d2d353534 0057 050B 01B1 41300000 09

Record id=39 is at byte location 84
39,Pharmacia and Upjohn Company,VB-517,68,1248,302,60.6,false,false,true,true
00000027 1C 506861726d6163696120616e642055706a6f686e20436f6d70616e79 56422d353137 0044 04E0 012E 42726666 03

Record id=817 is at byte location 134
817,Custom HBC Corporation,ON-989,55,1466,110,19.9,false,true,true,false
00000331 16 437573746f6d2048424320436f72706f726174696f6e 4f4e2d393839 0037 05BA 006E 419F3333 06

Java Bitwise Operator Tutorial

Attached Files: BitwiseTutorial.java (6.538 KB)

I wrote this small Java program to illustrate the use of bitwise logical operators to read or set individual bits in a byte.