

[Mark as done](#)

Opened: Sunday, 18 August 2024, 12:00 AM

Due: Saturday, 31 August 2024, 11:59 PM

Assignment 1: Build and test a dblayer on top of toydb

#CS631 Assignment.

This assignment is to be done in groups of 2. Please submit the assignment only once per group, with a tar.gz file named `rollnumber1_rollnumber2.tar.gz`. If you are stuck unable to find a partner, contact us as soon as possible.

This assignment has an attached tar file, see below.

You are given a toy database, called toydb, in the attached toydb.tar.gz file, with some missing parts that you have to fill in. This assignment is conceptually split into three tasks.

#1. Building a record layer

First, you need to fill in the missing code in the dblayer directory. It is a record or tuple layer on top of a physical layer library (pflayer, which is provided to you). The physical layer library presents a paged file abstraction, where a file is logically split into pages.

You have to structure each page as a **slotted-page structure**. That is, the **header** at the top of the page must contain the following information: an **array of pointers** (offset within the page) to each record, the **number of such records**, and the **pointer to the free space**. The actual record data is stored **bottom up** from the page. Tuples are addressed by a **4 byte 'rid'** (record id), where the first 3 bytes identify a page, and the other byte is the entry number in the slot header.

Note that this layer treats the record as a blob of bytes, and does not know about columns or fields.

For this part of the assignment, search for "UNIMPLEMENTED" in `tbl.c` and `tbl.h`, and put in the relevant code.

The structure of a record is created by the functions in `codec.c`. Encode and decode functions are provided for primitive types, which can be used to store and retrieve values to/from a byte buffer. However, the functions to create records, with variable length types, and to retrieve attribute values from records, must be implemented by you by filling in the "UNIMPLEMENTED" parts in functions in `loadddb.c`. Note that these functions have to be coded per relation, since our simple data storage system does not have metadata about relations; this could be a future extension.

#2. Testing: loading CSV data into db, and creating an index.

We will load up a table using the API above, from data contained in a CSV file. You have been supplied code in `loadddb.c` to do all the relevant parts; you just have to fill in the "UNIMPLEMENTED" parts.

The first line of the CSV file contains schema information; for example:

```
country:varchar, population:int, capital:varchar
```

The data type can be one of "varchar", "int", "long"; the maximum sizes of each field is assumed to be less than the page size, and further all the fields in a row together fit in a page.

The rough steps are as follows:

```
for each row in the csv file,  
  
    split it up into fields  
  
    encode each field (according to type) into one record buffer  
  
    rid = Table_Insert(record)  
  
    AM_InsertEntry(index field, rid)
```

The idea is to use the table API you just built, which returns a record id, then supply that record id to a BTree indexer. That code (with the prefix AM_, for access method) is made available to you. You simply have to read the docs.

#3. Testing: Retrieving the data.

Fill in code in `dumpdb.c`.


dumpdb has two ways to retrieve data (depending on a command-line argument).

"dumpdb s" does a sequential scan, implemented using Table_Scan

"dumpdb i" does an index scan. Use the AM_ methods to do a scan of the index, and for each record id, invoke Table_Get to fetch the record.

In both cases, you have to decode the record to print it back in the same format as the csv file, so that we can compare the original CSV file with the version reconstructed from the database. There should be no difference.

Miscellaneous details:

- 1. Familiarize yourself with the `am.pdf` and `pf.pdf` docs on the parts that are already built. You don't need to understand the internals though.
 - 2. Invoke `make` in each of the `pplayer` and `amlayer` directories before building the dblayer.
-  [toydb.tar.gz](#) 18 August 2024, 10:26 PM

Add submission

Submission status

Submission status	No submissions have been made yet
Grading status	Not graded
Time remaining	11 days 13 hours remaining
Last modified	-
Submission comments	► Comments (0)