UMass CS645 Lab1

Team members and contributions:

1. Po-Heng Shen 33953873
    - Makefile, Row, Page, LRUCache in Buffer Manager, Unittest, end2end test
2. Hsin-Yu Wen 33967467
    - Buffer Manager
3. Yanqi Chen 34540342
    - Fix errors in Buffer Manager
4. Tung Ngo 33297893
    - Unittest

Repo:

https://github.com/BoddyShen/UMass-CS645

How to run:

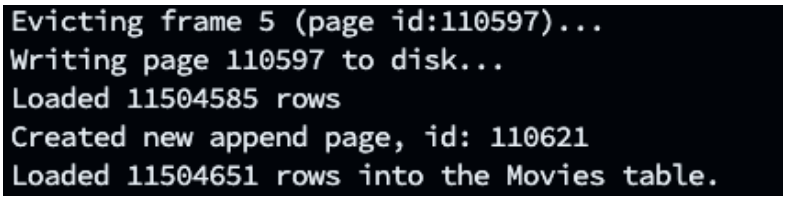
Please refer to the README of the repo.

1. Tested platforms: MacOS, Linux
2. Prerequisites:
    a. A C++ compiler (e.g., g++)
    b. GNU Make
    c. Add title.basics.tsv in `lab1` folder (Ref: https://datasets.imdbws.com/)
3. Build and Run
   This project uses a Makefile to manage the build process.
    - **Build the Project**
      To compile the project, enter each lab folder and run:
      `make`
    - **Run the Project**
      To run the project, `lab1` folder and run:
      `make run`
    - **Test the Project**
      To do unit test on the project in `test_unit` folder, enter each lab folder and run:
      `make unittest`
      To do end-to-end test on the project in `test_end2end` folder, enter each lab folder and run:
      `make end2endtest`
    - **Clean the Project**
      To remove the compiled files and the executable in build/, run:
      `make clean`

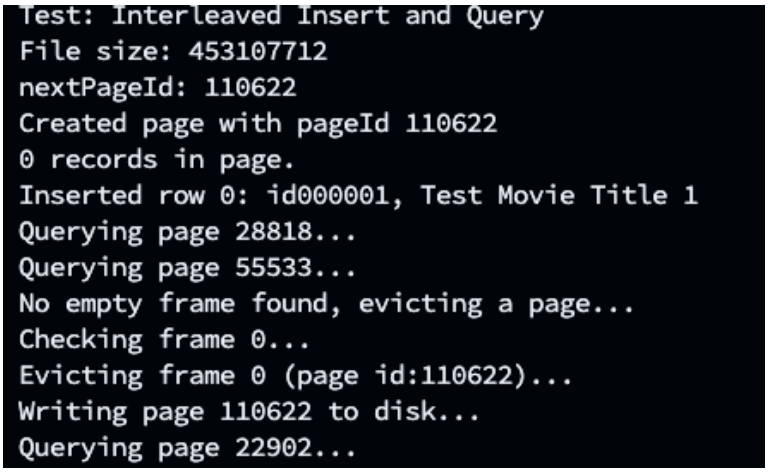
Design Choices:

1. Build system: Makefile
    - Makefile is the usual build system in c++ for users to define a set of rules to compile and link programs.
2. Buffer Manager
    - Implement the Least Recently Used (LRU) algorithm using a doubly-linked list (see LRUCache.cpp) for efficient page replacement. We maintain a node in the doubly-linked list for each frame in the buffer manager. And we use an unordered_map to keep track of node pointers for each frame. This allows us to move the corresponding frame's node to the end of the list in O(1) time whenever a row is inserted or a page is accessed, indicating that it was recently used.
    - We allocate a continuous space for bufferSize number of page frames. We use unordered_map to map pageIds to buffer pool frames.
    - We use a metadata struct to keep track of page ID, dirty status (if page gets modified) and pin counts.
    - Dirty pages will be written to disk during eviction. To ensure durability of data, all dirty pages will also be written to disk when the BufferManager is destructed.
3. Page
    - The first 4 bytes are used to keep track of the number of records in the page.
    - The rest of the page is used to store the record data. The i'th row is stored at offset 4 + i * RECORD_SIZE.
4. Test
    - Unittest:
      We perform unit tests for both the Page and Buffer Manager components to verify their implementations. In the Buffer Manager unit test, we initialize the Buffer Manager with a capacity of 2 frames, create 3 pages, and insert rows into them. This setup tests that the eviction procedure functions correctly when the buffer pool is full.

    - End-to-end test:
      This section covers both loading and querying.
      For loading, we read the movie ID and title from the "title.basics.tsv" file line by line. Each line is parsed into a Row object, which is then inserted into a page using the buffer manager. When a page becomes full, the buffer manager's eviction procedure is triggered, writing the evicted page to the data file.
      From the output below, we can observe there is 11504651 rows in movie table, and we create 110621 pages, 11504651/110621 is about 104 row per page, and is comply with the (4096 - 4) / (9+30) = 104.923, is also 104 row per page.



For querying, we follow the example provided in the lab instructions. First, we initialize the Buffer Manager with a capacity of 2 frames, create a new page, and insert several rows into that page. Then, we simulate querying random pages from the previously loaded database multiple times to trigger the eviction mechanism. Finally, we verify that the previously inserted rows remain correct after being evicted and reloaded by the Buffer Manager.
From the output below, we observe that after a new page with ID 110622 is created and a new row is inserted, and given that the buffer size is only 2, the second random query operation triggers a page eviction. Consequently, the least recently used page—page 110622—is evicted.



Assumptions:

- The page size is 4KB.
- Only consider a single table and fixed schema (Movies(movieId: char(9), title: char(30))).
- There are only insertions and no deletions.

Limitations:

- No waiting procedure when there is no unpinned frame in the buffer manager.
- Only support a single thread since we don’t introduce locks in the critical section.