CS 6530 Database Systems

Project 2: SimpleDB Basics (Buffer Replacement and B+ Tree Index)

Team 15 - Tanvi Gangadhar(u1205740), Greeshma Mahadeva Prasad (u1141804)

DECISION DESIGN

Page eviction for SimpleDB Buffer

- We used a stack/LIFO for the replacement policy in evictPage. Kept track of the pid's of pages in the BufferPool and evicted accordingly when the BufferPool was full. Called the evictPage method inside getPage method.
    - Push to the stack while getting a page
    - Pop from the stack when the bufferpool size is equal to the maximum number of pages
- flushPage method writes to the disk before the page is removed from the bufferpoolmap by evictPage method. Flushallpages calls flush page as when when all the pages need to be written to the disk.

Search in B+ Tree

- Assuming that findleafpage function will receive pages of type LEAF and INTERNAL, we had to recurse until the LEAF page was found. Set the permissions as specified in the requirements (READ_ONLY while recursing and the original permission passed to the method was set only for the LEAF node to be returned). The internal page to recurse on was decided based on the key value comparison.
- Implementation of IndexPredicate and Predicate was straightforward. We implemented the getter methods and the compare, toString and filter methods.

Insert in B+ Tree

- Initially we tried creating a new right page as suggested but we found implementing it by creating a new left page made it easier to keep track of the iterator for the new parent entry. Updated the children and sibling nodes of all the affected nodes.
- Once splitLeafPage was done, implementing splitInternalPage method was much quicker since it had a similar structure.

Delete in B+ Tree

- These components were not very similar in implementation unlike the insert component. For steal methods, we computed the number of entries/ tuples to move for the redistribution and updated the children and siblings accordingly. Iterated through the nodes until the tuples/b tree entries were evenly distributed.
- Merge was comparatively simpler, it involved deletion of the parent entry, updation of sibling pointers and making the right page available for reuse using the setEmptyPage method.

API CHANGES

We have not made any changes to the provided API. We did not have to implement any new classes for this project.

MISSING/INCOMPLETE ELEMENTS

All the unit and system test cases passed, we believe we have implemented everything needed as per project 2 requirements.

TIME SPENT

Similar to proj1, we pair programmed most of part of the project and together we spent about 20 hours over a few days. The breakup is as shown below,

- Understanding the provided methods such as updateParentPointers, getEmptyParentSlot etc and the structure of B+ Tree- around 3 hours
- Search in B+ Tree- 3 hours
- Insert in B+ Tree - 5 hours
- Delete in B+ Tree - 5 hour

DIFFICULTIES ENCOUNTERED

While implementing evictPages() and we changed the replacement policy a couple to times as there were errors. Some parts of this project consumed as lot of time when the system tests were failing. Even though our unit tests passed for insert methods, system tests would fail. It took a while to figure out we needed to implement insertTuple. Similarly delete methods' system tests would fail as there was a flaw in our deleteTuple implementation.