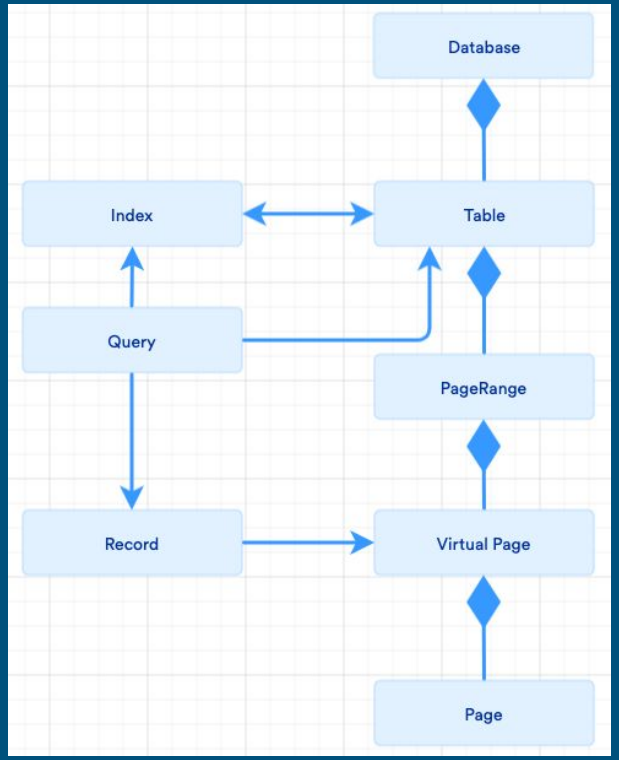
Milestone 2

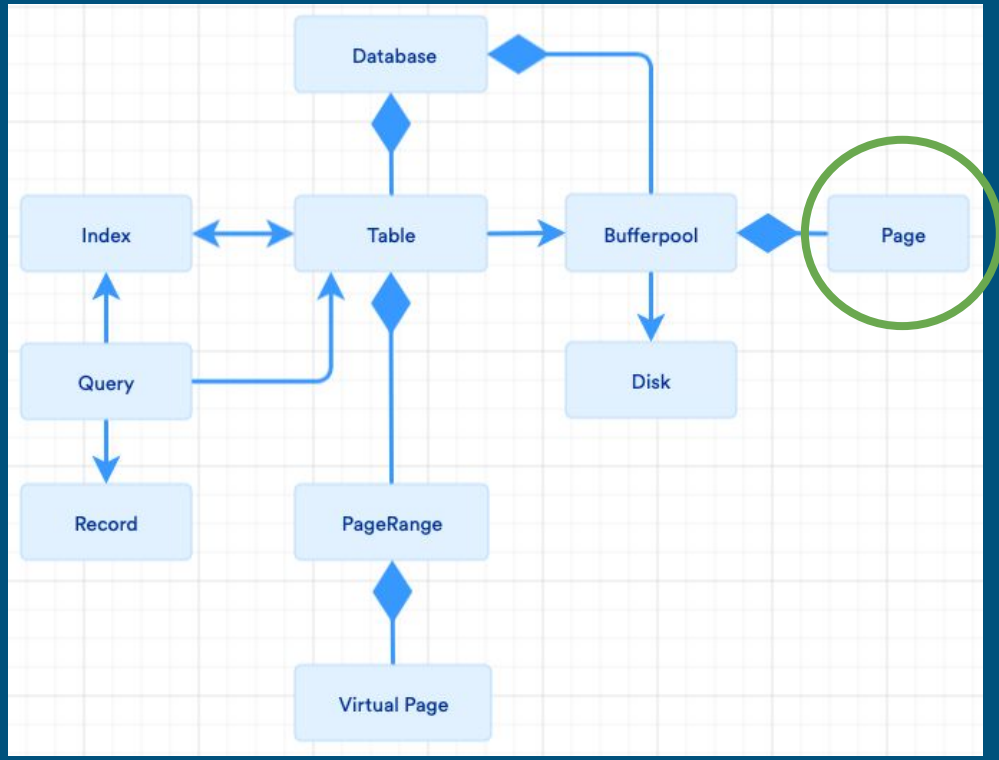
Merge and Persistent Store

Shivani Parekh, Soumya Durisetti, Alvin Agana, Daniel Silva, Christopher Pires

Milestone 1



Milestone 2



Bufferpool : New Classes

Disk - writes and reads files in a directory specified by database

- One file per physical page
- File name contains identifying information about page

Bufferpool - represents main memory

- Stores Page objects
- Eviction policy based on Clock
- Keeps track of dirty pages and pin counts
- Returns page object to caller (Table class) based on page information
- Gets page from disk if needed

Bufferpool: Changes to M1

Table class

- Manages pages - creates Pages and requests them from Bufferpool
- When pages are initially created, inserted into Disk
- Calls on bufferpool to get pages from memory or disk, handles pin/unpin

VirtualPage class

- Stores list of page locations instead of page objects

Page class

- Has a page.location attribute, which is
  - (table_name, page_range_id, virtual_page_id, page_id/column)

Bufferpool: Changes to M1

DB class

- Created a bufferpool instance shared across the entire DB
- open()
  - Repopulates tables from disk if any
  - Sets path for bufferpool/disk
  - Repopulates physical page locations using file name structure
    ex: “Grades-0-B_0-4.txt”
- close()
  - All dirty pages in bufferpool are written to disk
  - Table information (page directories, RID directories, Index directory - JSON) are written to disk

Merge

Called in update() query for every 100 updates to a base page

- Creates Background-thread that consolidates base pages
- Main-thread continues to handle queries
- Uses Python’s Threading Library

Merge - Implementation

Main-thread

- Default thread which handles regular flow of execution (eg. queries)
- Updates page directory before the start of the next merge process
  - Consolidated base page stored as an attribute of old base page

Merge-thread (background)

- Consolidates new updates from TID to TPS into a deep copy of the base page
- Once updates are made, background-thread merges with the main-thread

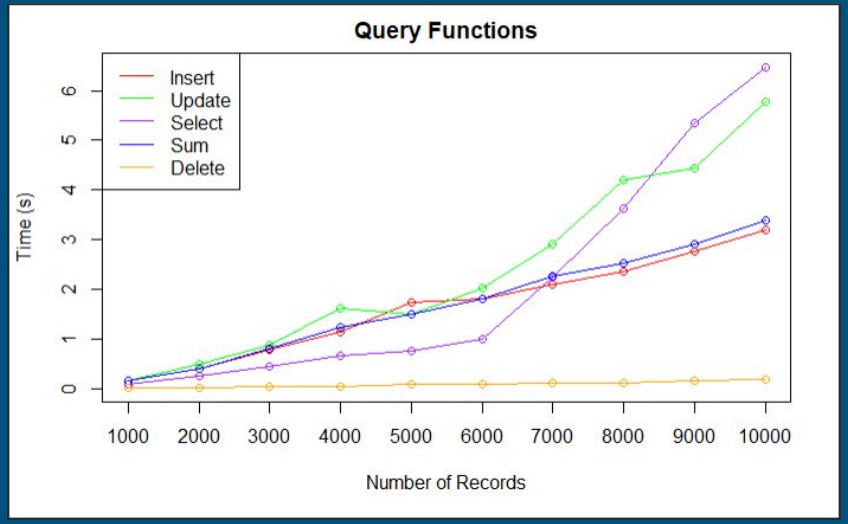
Index

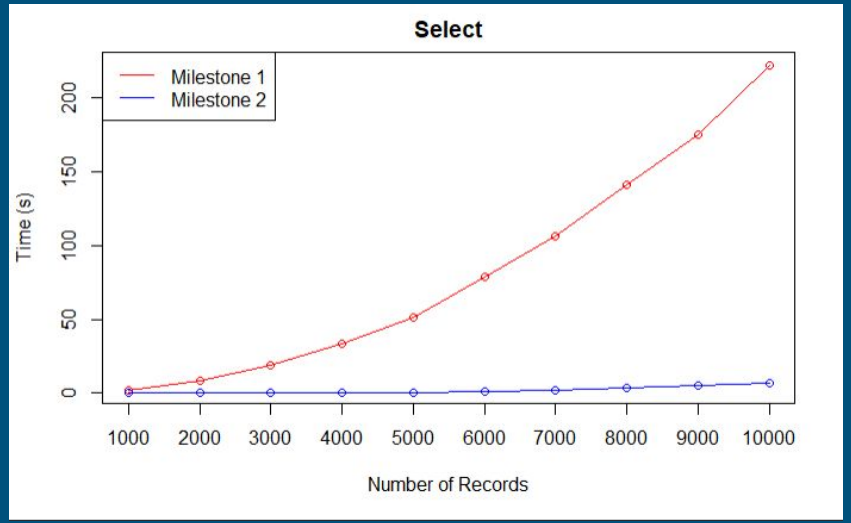
- Implemented as a Hashtable
  - Key is data value
  - Value is list of rids (just one rid for index on primary key)
  - Primary key always has an index
  - Indexes can be created and dropped on different columns
  - One hashtable per column
- Pros
  - Quick access time for single values
  - Easy implementation
- Cons
  - Takes longer to access a range

Coding Quality Improvements from Milestone 1

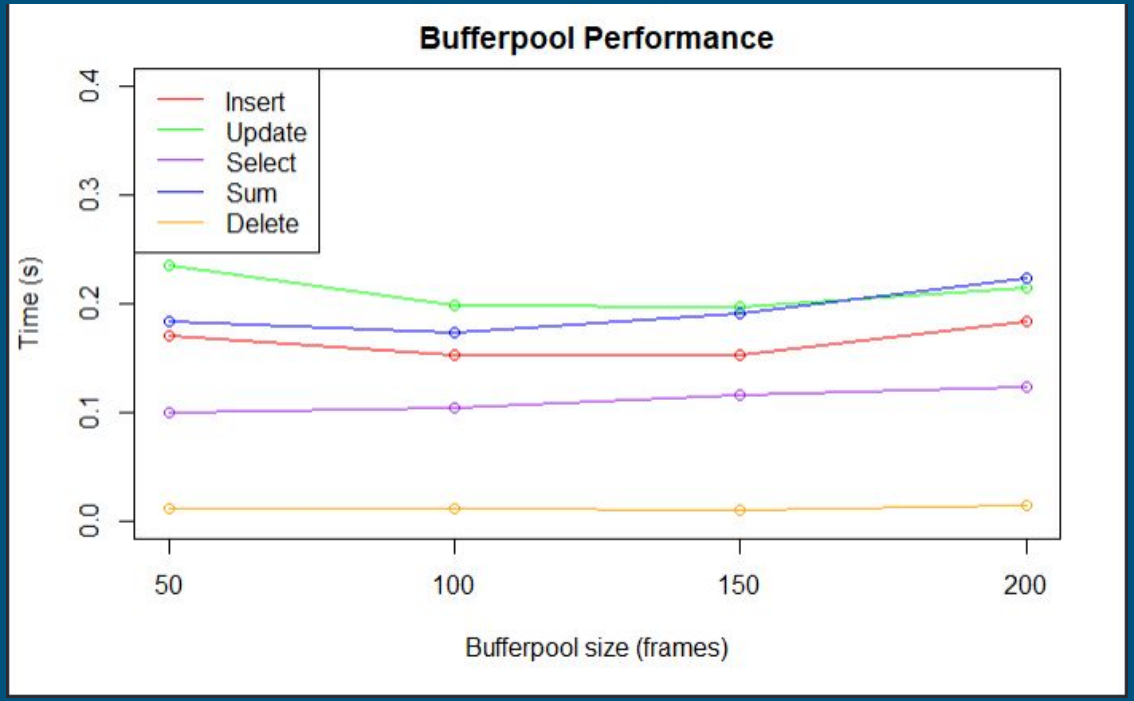
- Unit testing + end-to-end tests: thoroughly test out codebase
- Input validation: makes code more robust and easy to debug
- Standardized function structure and commenting etiquette

Results


Query Functions
Insert
Update
Select
Sum
Delete
Time (s)
0
1
2
3
4
5
6
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Number of Records


Select
Milestone 1
Milestone 2
Time (s)
0
50
100
150
200
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Number of Records

Results


Bufferpool Performance
Insert
Update
Select
Sum
Delete
Time (s)
0.0
0.1
0.2
0.3
0.4
50
100
150
200
Bufferpool size (frames)

Future Improvements

- Optimize merge threshold for better performance
- Create more unit tests for easier debugging in the future
- Break large functions into smaller methods for improved readability

Challenges

Initially, we struggled to understand how to implement bufferpool and merge.

- Merge
  - Where and when to call merge function
  - How to have a thread merging records in the background
- Bufferpool
  - How to represent disk/ what to put in the files
  - How to re-populate tables from persistent store

Takeaways

- Better understanding of how persistent store works
  - Didn’t realize that not only page data gets stored but also page directory
- When splitting up tasks we should discuss changes/ plan of action together
  - Maybe have a project coordinator role