Milestone Two

Nick Abcarius
Andrew Do
Travis Garcia
Nicole Pavlovich
Steven Tan

UC Davis, ECS 165A
Prof Mohammad Sadoghi
Milestone Goals

- Implement a buffer pool that interacts with disk storage to store and retrieve data
- Add a merge function that periodically merges tail data with base data
- Create an index class that allows the user to index on any column for query functions
Bufferpool
Bufferpool

The bufferpool stores database pages that are actively being manipulated

- Main memory is faster to access than disk
- However, information in the bufferpool is lost in the event of a crash
- Therefore, all changes must be eventually copied back to disk

Queries ask for pages, which are returned from the bufferpool or acquired from disk if not already available

**Pinned Pages:** Pages currently in use by a transaction
Bufferpool: Eviction

If the bufferpool is full and a page stored on disk is requested, eviction must occur.

**Dirty Page:** A page in the bufferpool with changes not reflected in the disk.

**Eviction Policy:** Least Recently Used, Page-level granularity

- Each page is assigned an age that is updated each time the bufferpool is accessed.
- The **oldest** page is evicted when a new page is needed.

```python
# page.py

def writePageToDisk(self, path):

def readPageFromDisk(self, path):

def writeToDisk(self, path) # physical page

def readFromDisk(self, path) # physical page
```
Merge
Merge Design

**Purpose:** Speeds up queries

**Key Design:** Asynchronous, lazy merging does not interrupt transactions and single pointer swaps ensures minimum contention

**baseRID:** Tail records store the baseRID in their metaColumns

**Granularity:** Global MergePolicy which defines how many tail pages inside a PageRange will be filled before a merge is initiated

**Tail Page Sequence Number (TPS):** Number per base page that tracks RID of the last tail record merged.
Merging Process

1. Create copy of base pages in a page range.

2. Iterate over tail records in filled pages.
   Use metacolumn baseRID to find correlating base page.

3. Copy records if safe to merge, update base page with tail values.

4. When merge is finished, replace the old base pages.

Note: After merge, if indirection value is less than TPS, record has been merged so return consolidated base record.
Indexing
Indexing

- Indexing is used to more effectively find and select records.

- The user can create an index on any column by using:
  ```python
def create_index(self, column_number)
  ```

- In this example, an Index on column B would allow the user to select (3) and would return both record 01 and 02.

- Previous iterations of index used a dictionary, and then a binary search tree, before settling on B-Tree.
Structure: Order 3 B-Tree

- Sortable and Self-balancing
- Improved search time (find is $\lceil \log_2 N \rceil$ comparisons)

Nodes:
- Can be found by search key
- Contain RIDs of matching records

Cons:
- All internal and leaf nodes have data pointers, unlike B+ Tree
- Leaf and non-leaf nodes are of different size
- Deletion may occur in a non-leaf node
Milestone Two

Nick Abcarius
Andrew Do
Travis Garcia
Nicole Pavlovich
Steven Tan
Next Steps:

- Test & debug with tools such as Python cProfiler to analyze functions for optimization
- Implement concurrency, multithreading, and other ACID guarantees