ECS165A
Milestone 1 Overview

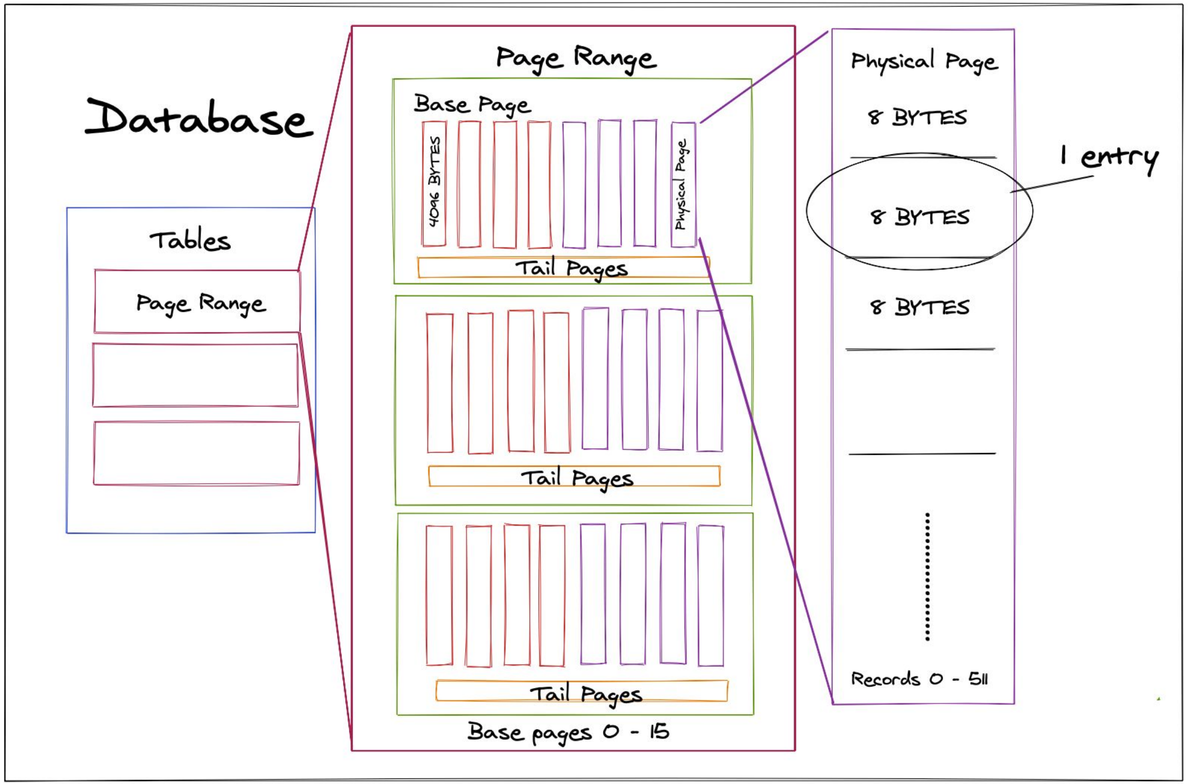
Haley Raizes, Brittany Bates,
Jim McKerney, Nicholas Chen, Chris Bried

Objectives

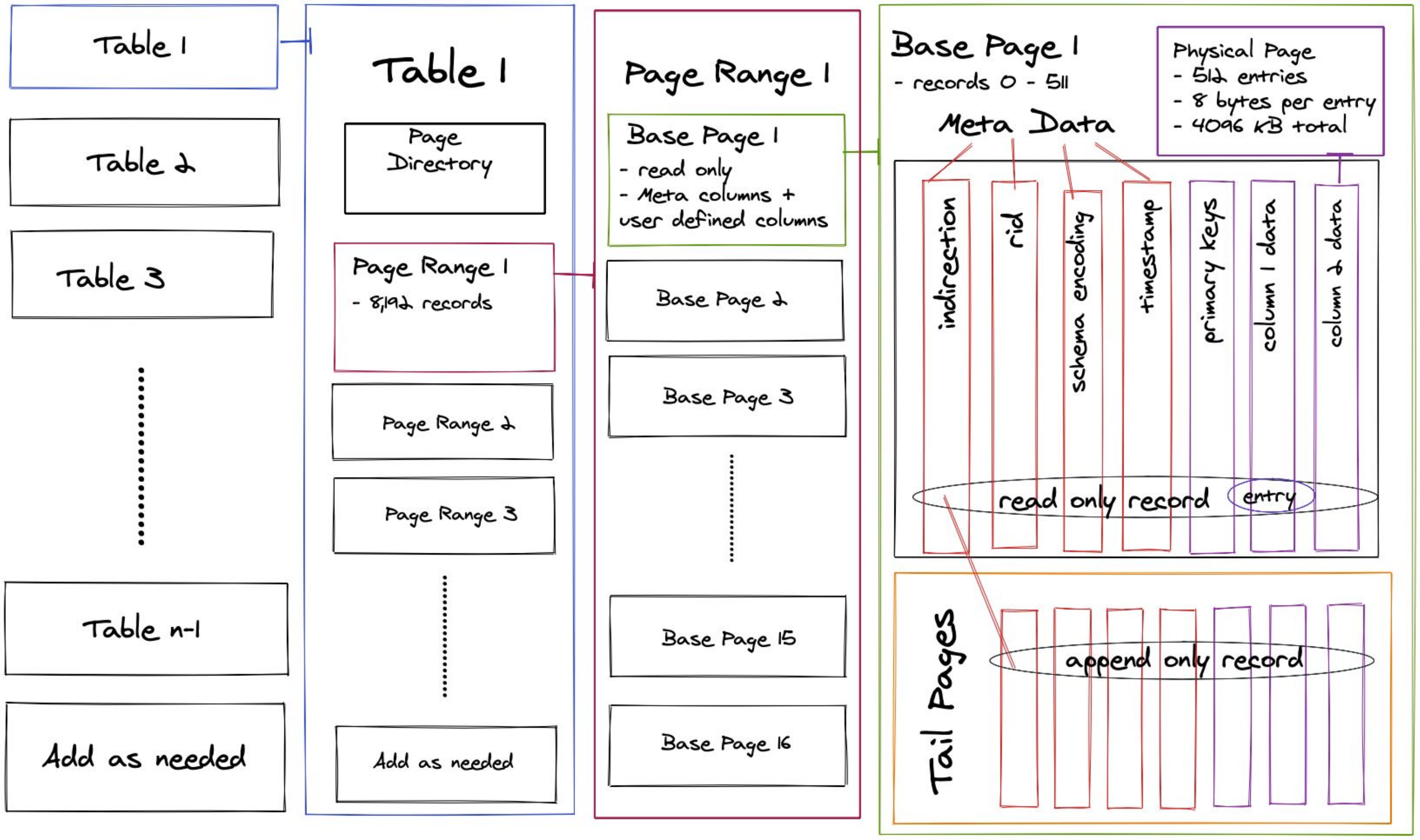
- Data Model
- Bufferpool
- Query API
- Performance
- Questions
- Demo

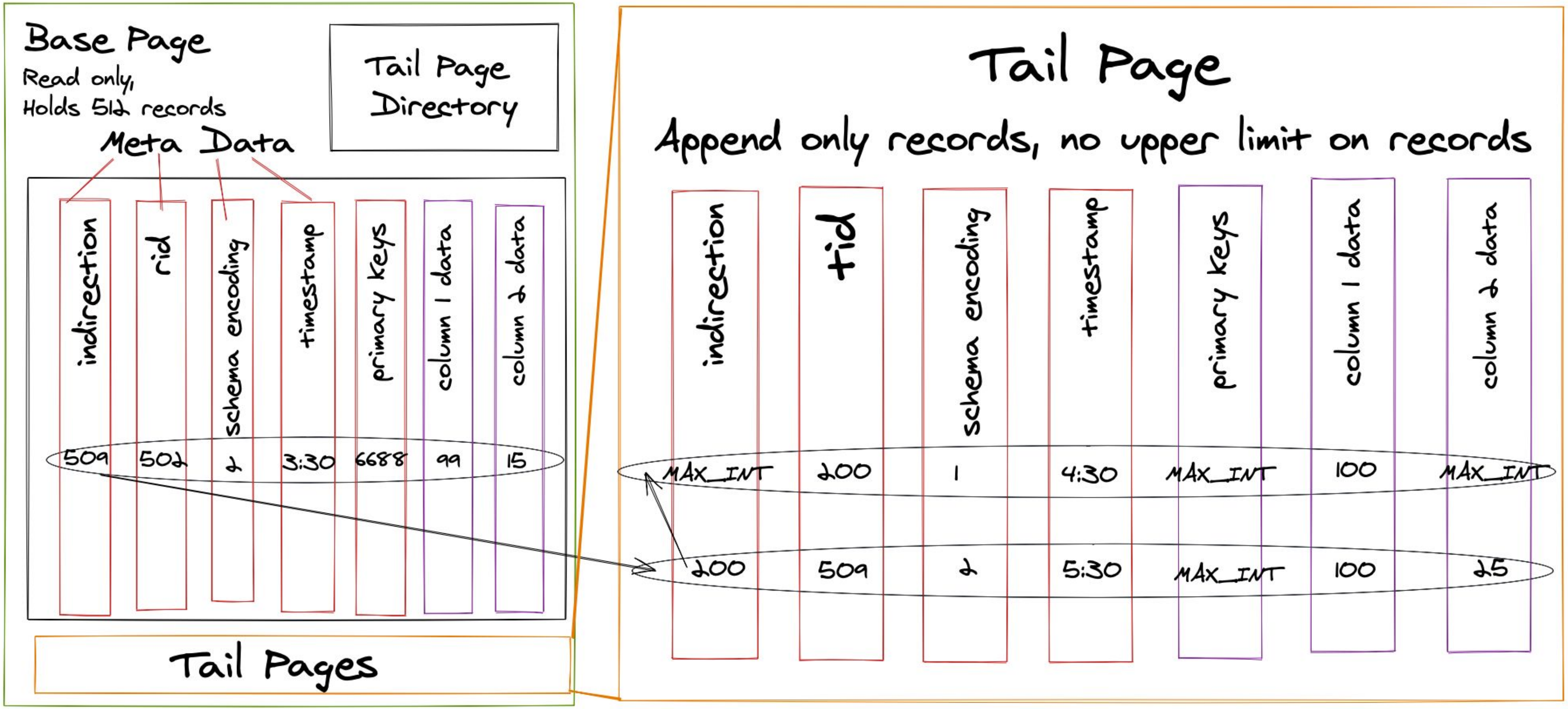
Lineage Based Data Store

A relational columnar database designed to bridge the gap between OLTP and OLAP workloads.



L-Store Database

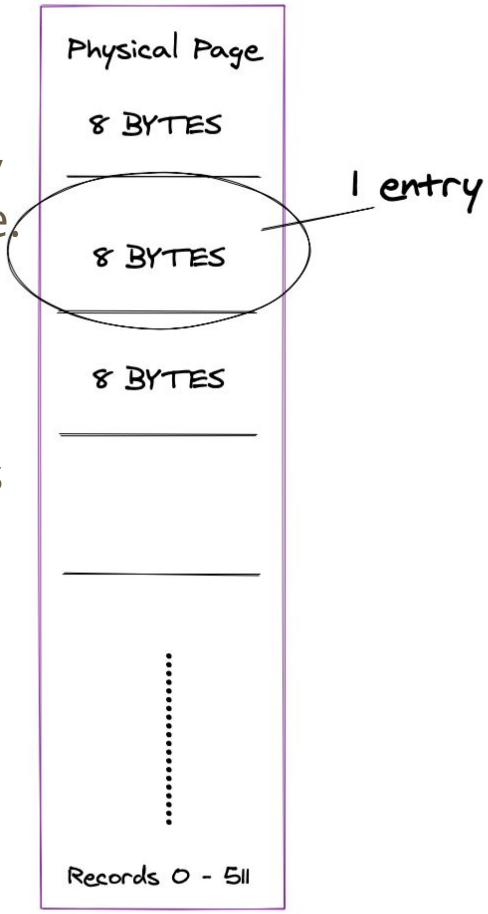



Base Page
Read only,
Holds 512 records
Tail Page Directory
Meta Data
indirection
rid
schema encoding
timestamp
primary keys
column 1 data
column 2 data
509
502
2
3:30
6688
99
15
Tail Pages
Tail Page
Append only records, no upper limit on records
indirection
tid
schema encoding
timestamp
primary keys
column 1 data
column 2 data
MAX_INT
200
1
4:30
MAX_INT
100
MAX_INT
200
509
2
5:30
MAX_INT
100
25

Physical Page Format

Finding an entry in the physical page:

- Physical page = bytearray of 4096 bytes, each column entry taking 8 bytes. This makes for 512 entries per physical page.
  - Bytearray size and record size can be configured as constants
- **Starting byte index**: row * PAGE_RECORD_SIZE
- **Ending byte index:** starting point + PAGE_RECORD_SIZE
- Writes positive integers only. Conversion into other data types (timestamp, None, string) is handled at the table level.
  - Write converts integer into 8 bytes
  - Read converts bytes back into integers
  - Special null value: (2^64)-1



Bufferpool

Page Directory maps RIDs to their corresponding page range, base page, and physical page index. RIDs are assigned using record count.

Page_Directory = { 64803: { “page_range”: 7, “base_page”: 14, “page_index”: 291}}

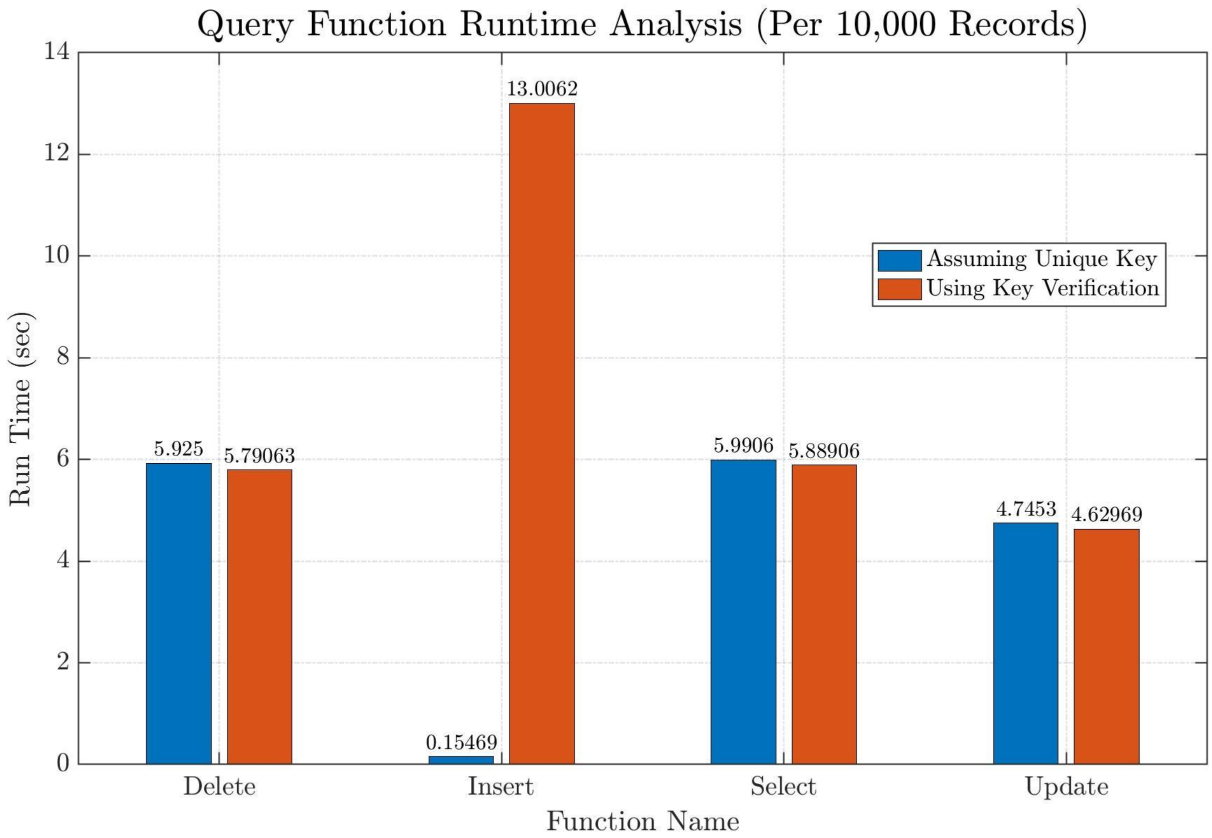
```python
def __rid_to_page_location(self, rid: int) -> dict:
    '''
    Helper function that returns a dict of the memory location for a given RID
    '''

    page_range_index = math.floor(rid / ENTRIES_PER_PAGE_RANGE)
    index = rid % ENTRIES_PER_PAGE_RANGE
    base_page_index = math.floor(index / ENTRIES_PER_PAGE)
    physical_page_index = index % ENTRIES_PER_PAGE

    return { 'page_range': page_range_index, 'base_page': base_page_index, 'page_index': physical_page_index }
```

Query API

Insert	Update	Select	Delete	Sum
1. Validate user args 2. Make new RID based on record count, set schema to 0, create new record object 3. Add RID to page directory and write the record	1. Check record exists and validate user args 2. Get the most updated record 3. Update schema encoding and add new updated values to columns 4. Create a new TID and record, set indirection 5. Write to tail page, change indirection and schema of base page	1. Check record exists and validate user args 2. Search for the key based on given column index and return the RID for any matches 3. Get the most recent updated record using RID 4. Keep selected values and replace the rest with None 5. Return a list of records	1. Check record exists based on the key 2. If exists, return the RID for that record 3. Set the “Deleted” value in our page directory to True 4. Future calls to this record will return False for this key	1. Loop through all keys and check if the key is between the start range and end range 2. If between ranges, add the value at the aggregate column index to the running sum 3. Return the sum or False if no records fall between the given range

Optimizations & Performance

- RID to page location is O(1)
- Validate user arguments before writing a record
- Integers for schema encoding



*These times are based on 10 run averages using the provided __main__.py

Processor: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, 4001 Mhz, 4 Core(s), 8 Logical Processor(s)

Questions

Demo