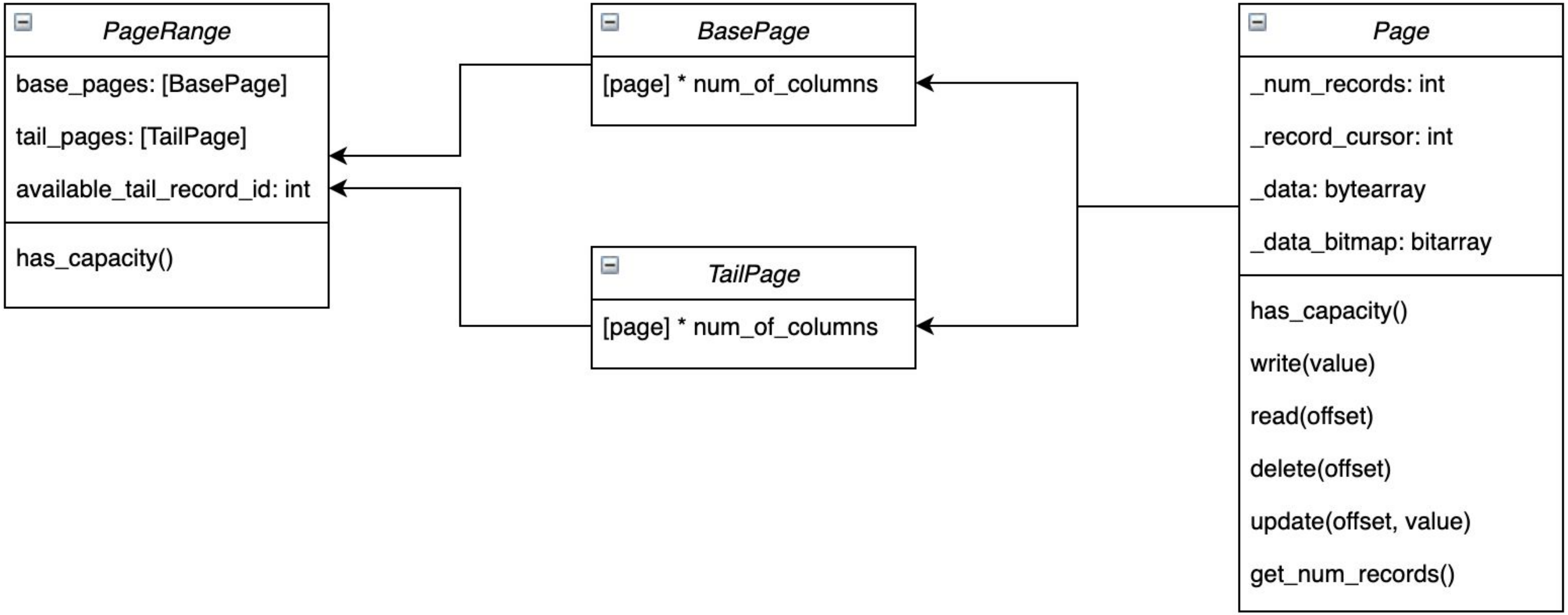
Froyo - Milestone 1
Single-threaded, In-memory L-Store

Developers: Heming Ma, Brian Lai, Roy Yi, Ruyi Yang, Yongxin Xu

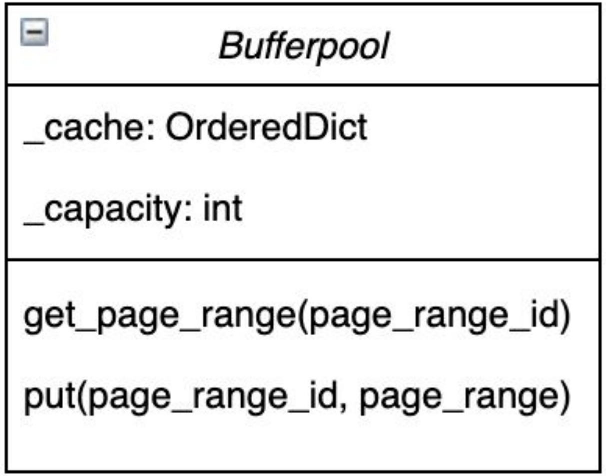
Schedule

1. Design and implementation (8 min)
2. Demo (4 min)
3. Q/A (8 min)

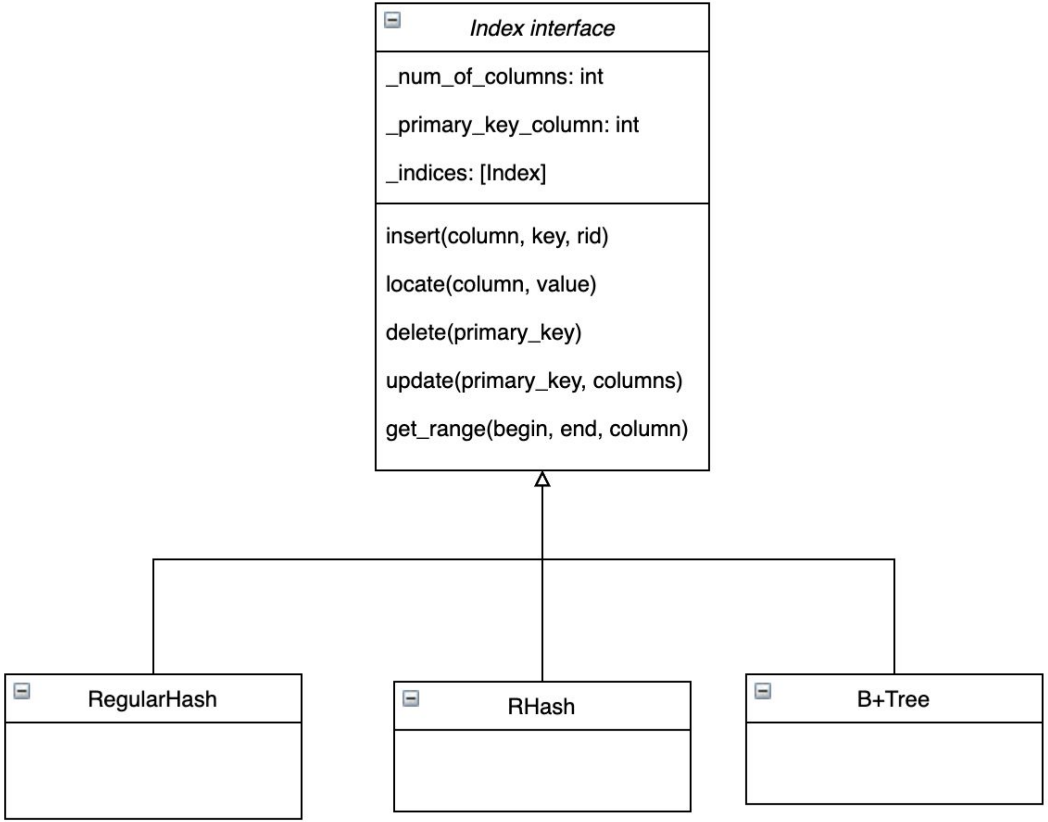
Page


PageRange
base_pages: [BasePage]
tail_pages: [TailPage]
available_tail_record_id: int
has_capacity()
BasePage
[page] * num_of_columns
TailPage
[page] * num_of_columns
Page
_num_records: int
_record_cursor: int
_data: bytearray
_data_bitmap: bitarray
has_capacity()
write(value)
read(offset)
delete(offset)
update(offset, value)
get_num_records()

Bufferpool


Bufferpool
_cache: OrderedDict
_capacity: int
get_page_range(page_range_id)
put(page_range_id, page_range)

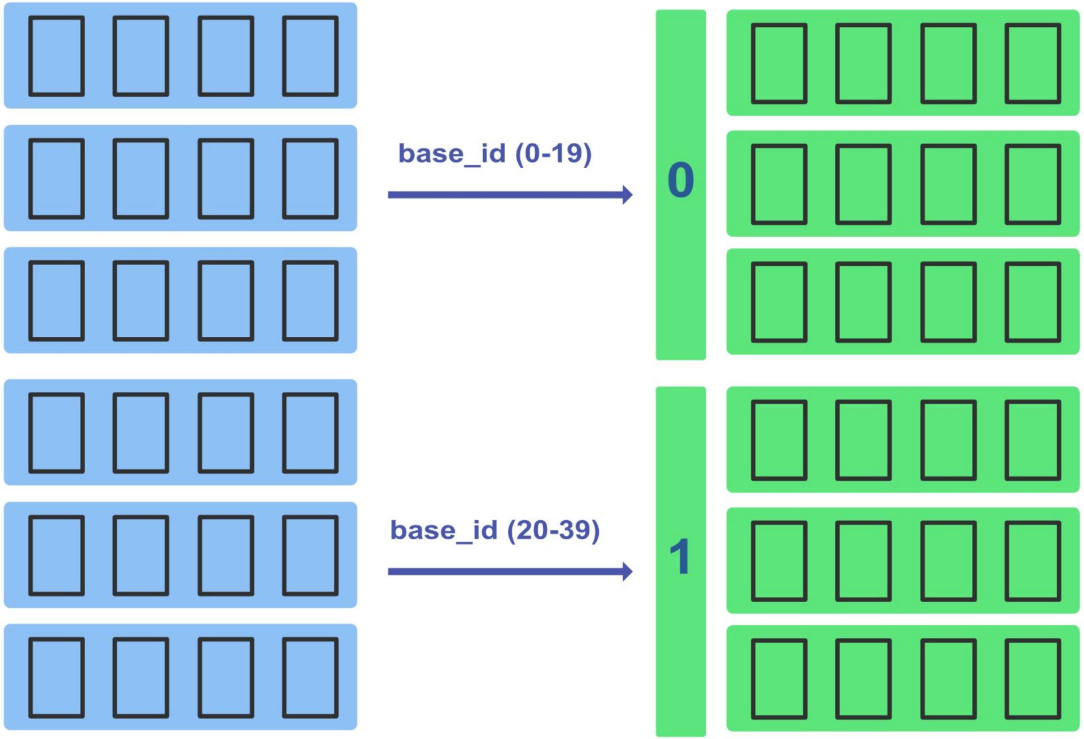
Index


Index interface
_num_of_columns: int
_primary_key_column: int
_indices: [Index]
insert(column, key, rid)
locate(column, value)
delete(primary_key)
update(primary_key, columns)
get_range(begin, end, column)
RegularHash
RHash
B+Tree

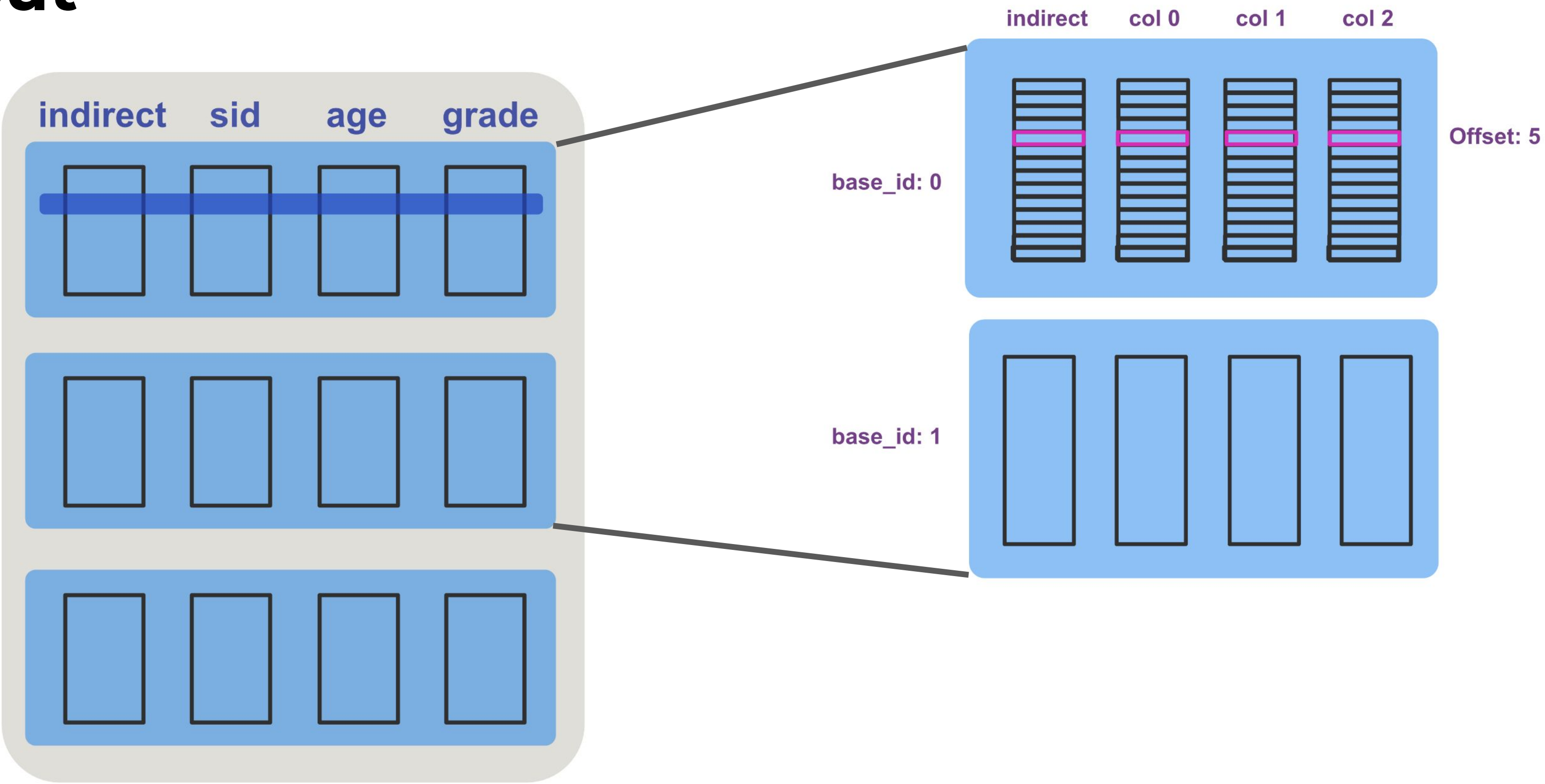
Table

Table
name: string _num_of_columns: int _primary_key_column: int _available_record_id: int _index: Index _page_ranges: [PageRange] _bufferpool: Bufferpool
insert(columns) select(index_key, column, query_columns) delete(primary_key) update(primary_key, columns) get_sum()

Page Range



Layout


indirect
sid
age
grade
indirect
col 0
col 1
col 2
Offset: 5
base_id: 0
base_id: 1

Query Operations

Insert: insert new entry in last base page

Delete: set indirection column to invalid flag

Update: insert new entry in last tail page

point base page indirection column to latest tail entry

Process_rid: index base entry with rid, go to latest tail entry if previously updated

Select/sum: scan through all rids in table to find all matches (using process_rid)

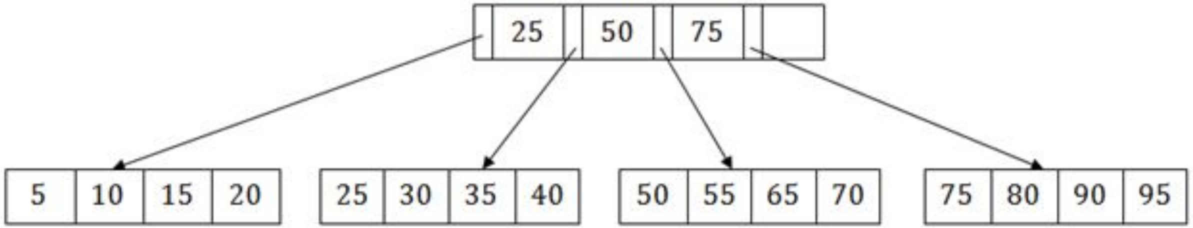
Index

- B+ tree
- Hash Table (Open-addressing)
- Hash Table (Separate Chaining)
- R-Hash Table (Open-addressing)
- R-Hash Table (Separate Chaining)

HashTable & B-tree Comparison

	B+ tree	Hash-table
Search operation	root-to-leaf O(logn) Sequential search available (sorted container)	Single I/O per look-up O(1)
Memory management	Easy memory allocation/deallocation (split node / merge node)	Hard to find an algorithm that expand/shrink the table size (rehash / open-address)
Insertion /deletion	O(nlogn) for batch insertion/deletion Have to maintain self-balance	O(1) for deletion and insertion O(n) for batch operation

B-tree Implementation



Challenge:

batch insertion takes O(nlogn)

Future Improvement:

Split node strategies

Size of b+ tree

(reference: https://www.programiz.com/dsa/b-plus-tree)

B+ tree performance

```
yruy0705@ad3.ucdavis.edu@pc19:~/ecs165/test1/ECS165A-Milestone1$ python3 __main
__.py
Inserting 10k records took:                          0.179706857
Updating 10k records took:                           0.271136593
Selecting 10k records took:                          0.1296896100000007
Aggregate 10k of 100 record batch took:  0.034964542999999996
Deleting 10k records took:                           0.050827964
```

Hash Table Implementation

We employ two types of strategies, separate chaining and open-address, to experiment the efficiency, with built-in Python hash function.

Open-Address (Using Python dictionary):

- No collision, meaning each entry only hold one slot.
- Insertion is more expensive for finding in an empty entry and allocating more space.
- Random probing algorithm.
- Selection in O(1)

Separate-Chaining:

- With constant table size and collision.
- Insertion in O(1)
- Selection in O(n/k), in the range between O(1) and O(n/k).

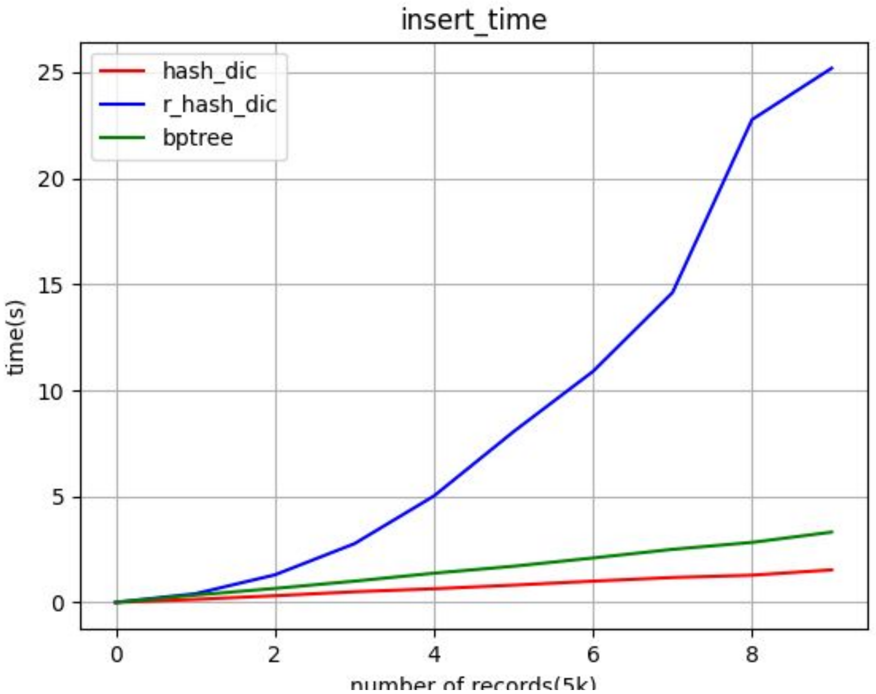
R-Hash Table Implementation

R-HashTable, differing from HashTable, provides linklist between each index in a sequential order that optimizes range queries.

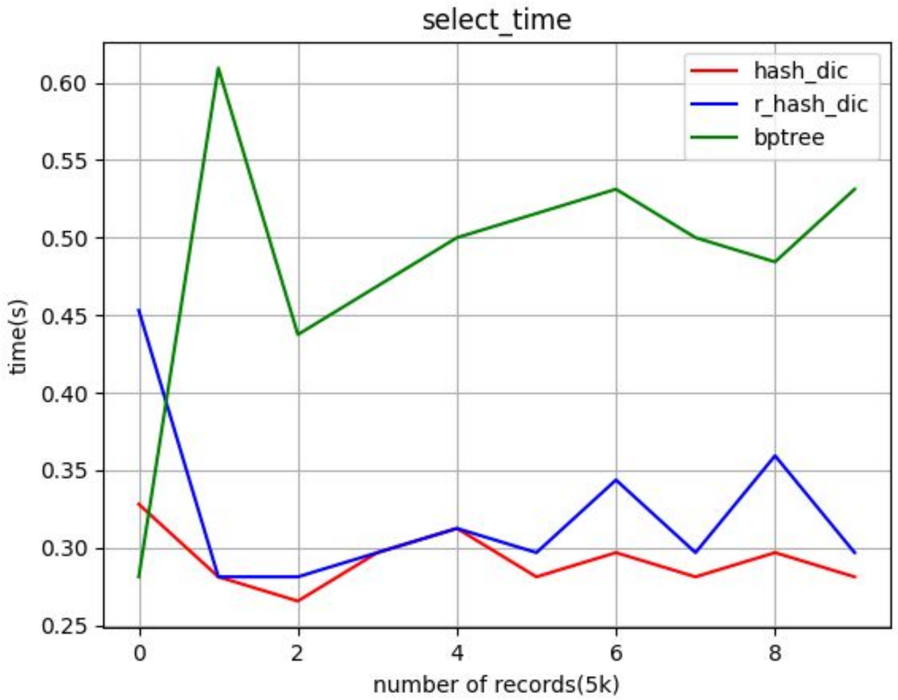
Comparing to Hash Table, R-Hash Table:

- Has slower insertion, update, and deletion due to the maintenance of sorted keys, which is O(n logn).
- Uses seeds to shorten range searches.
- Speeds up range queries in O(n/k), comparing to O(n) by Hash Table, where k is the number of seeds.
- Is Read-optimized and update-Heavy.

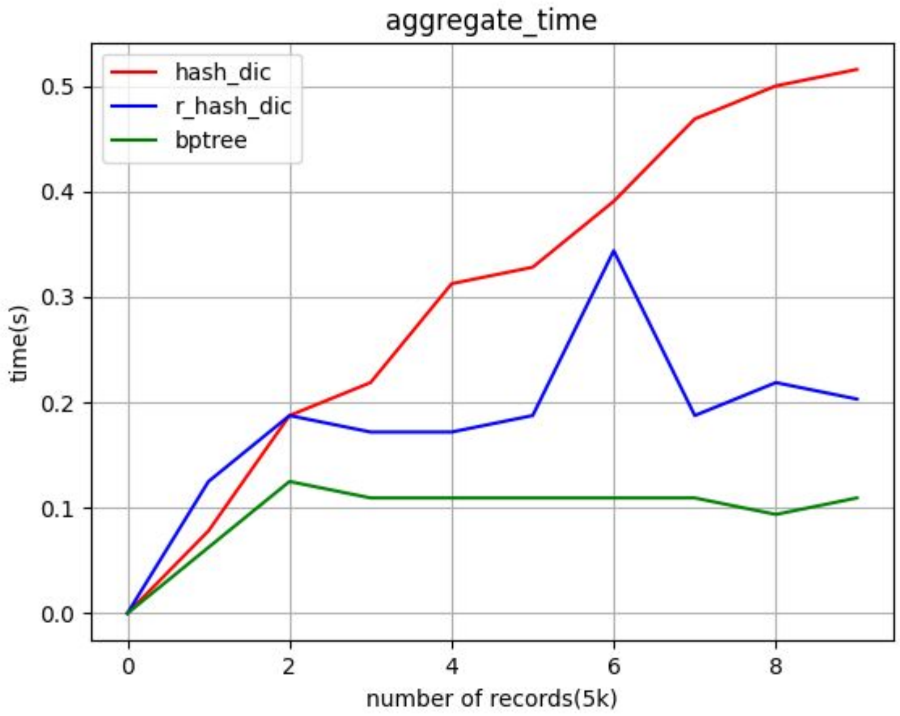
Performance for Insertion



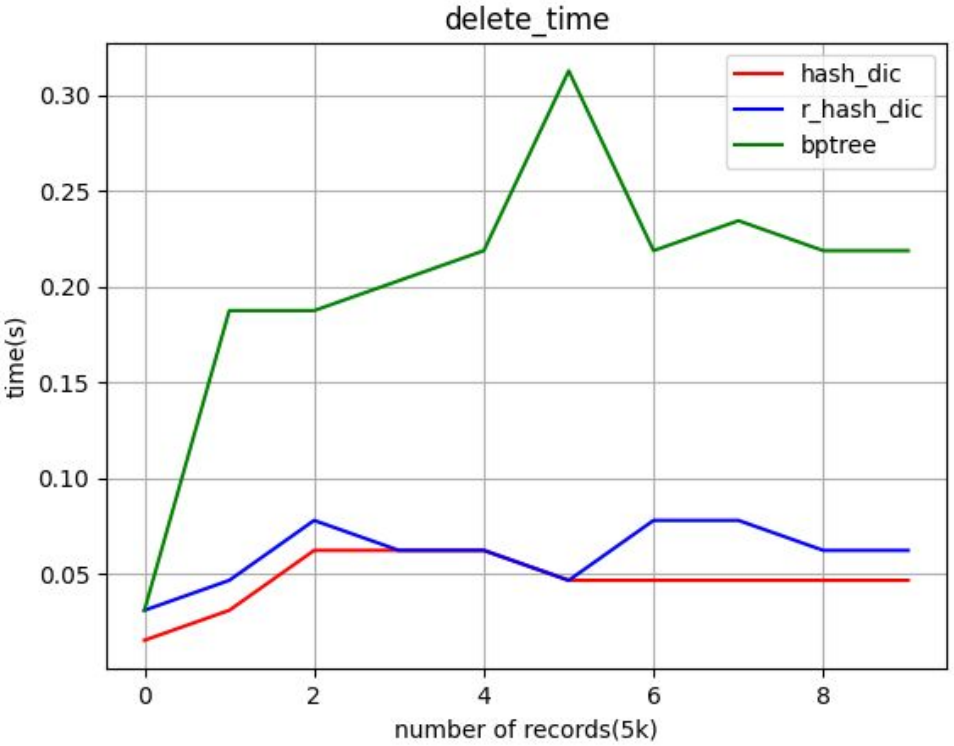
Performance for Selection



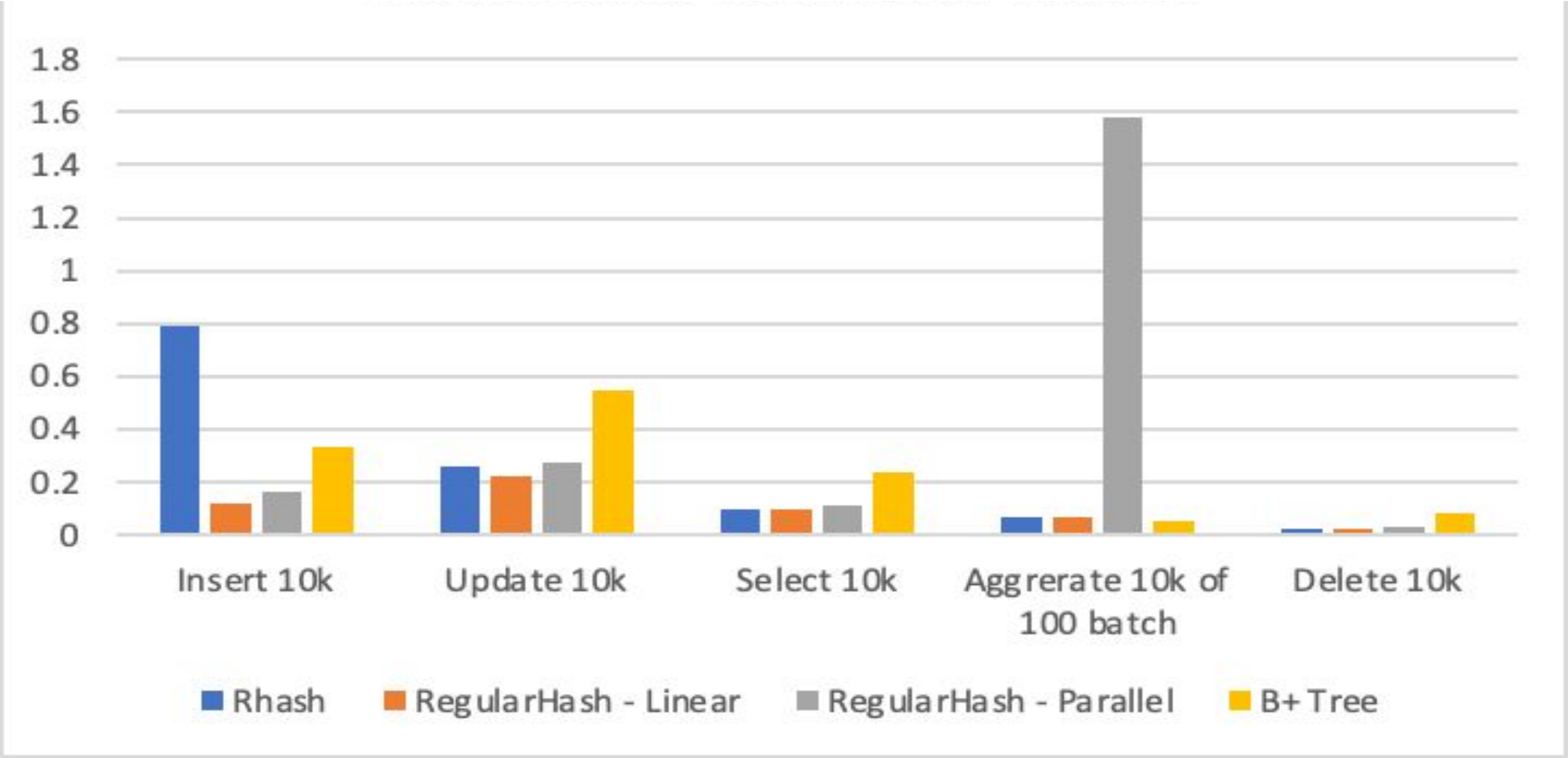
Performance for Range Selection



Performance for Deletion



Performance for Different Indexes



Questions & Demo