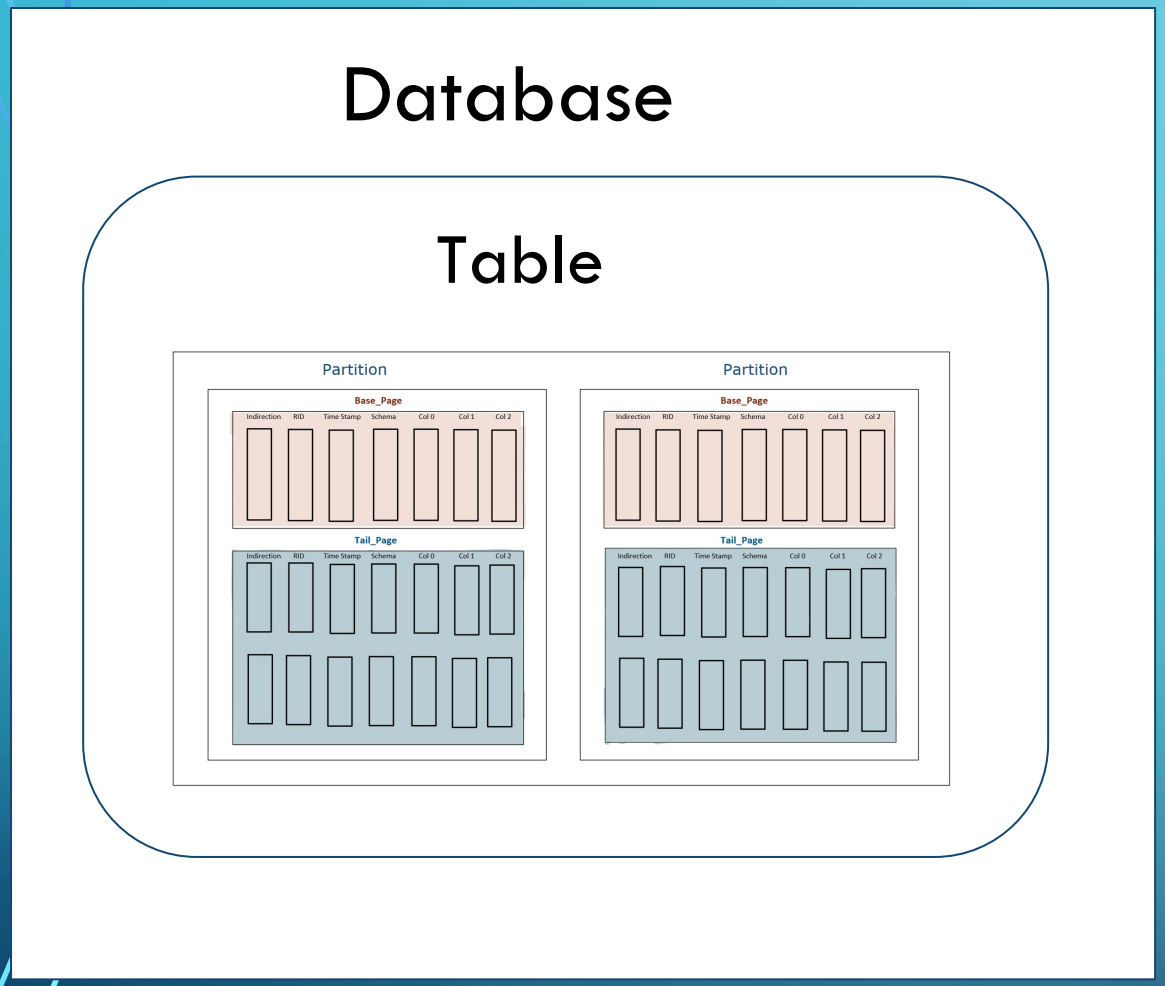
GROUP 4

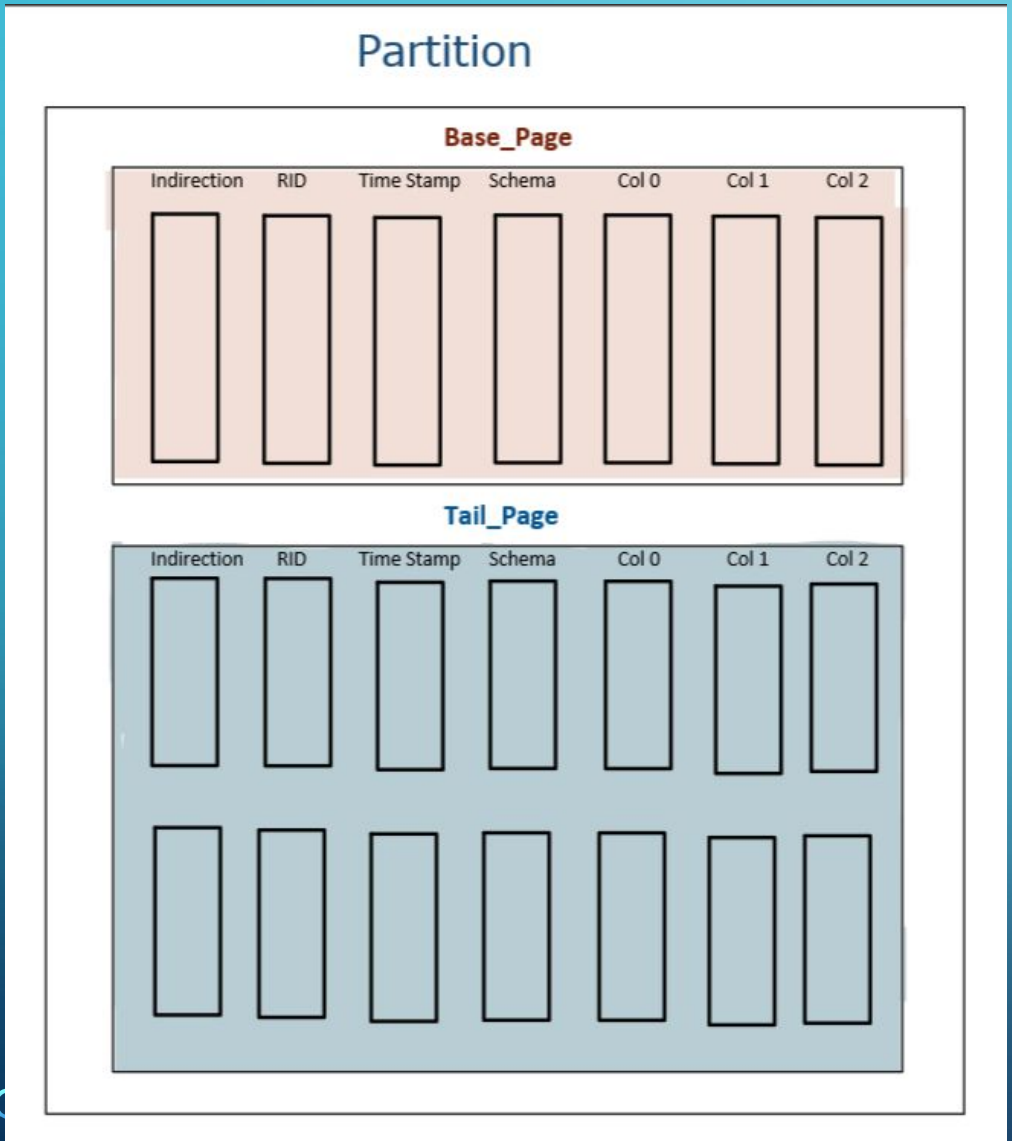
ANDY ZHU, MATTHEW DENITZ, SOPHIA IBRAHIM, TIMOTHY ZHANG, & XIAOWEI MIN

Overall Structure



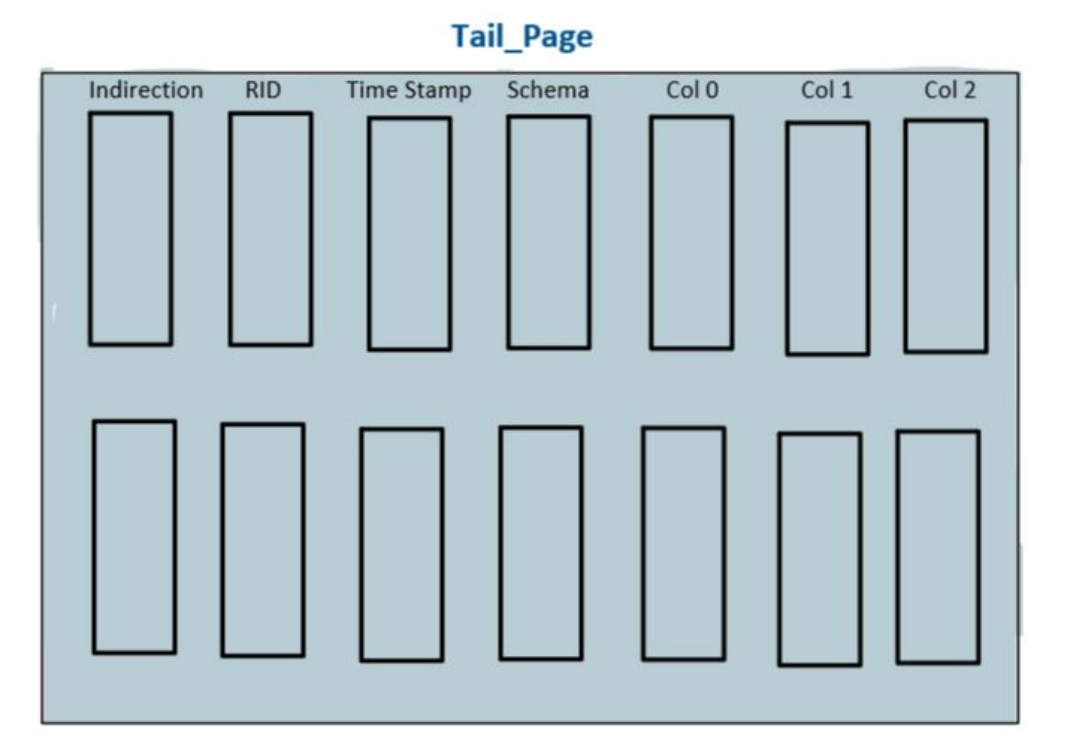
- Database Stores Table Objects stored uniquely by their name. These tables hold collections of Data I.E Student_Information,
- Each Table Holds a Partition List.
- We decided to implement a form of page ranges, called partitions. These partitions are stored in a partition list located in our Table object

Partition Structure



- Each partition holds a 1-D list of base pages and a 2-D list of tail pages.
- The base page (list) holds a page object for every stored value. The base page's purpose is to hold the initially inputted record data, and reference the indirection value for tail page traceback/lookup
- When the base page list (abstraction of l-store base page) becomes full (512 records/unique rows), we append a new partition to our partition list

Tail Page Structure



- Each element of the 2-D tail page list, is a list of physical tail pages which represent updated user records.
- Like the base page (a tail page stores at maximum 512 records spread across page objects.
- However, we allow more than 512 updates per base page by appending an additional tail pages to our tail page list

Overall Structure

- Records are partitioned into disjoint Page Ranges
  - A page range consists of a set of base pages
  - Each base page or tail page contains a set of physical pages, one for each column
  - Each page range consists of a set of tail pages.

Index

Table.py

def index(self, key, first_only=TRUE)

Partition.py

def index(self, key, first_only=TRUE)

Page.py

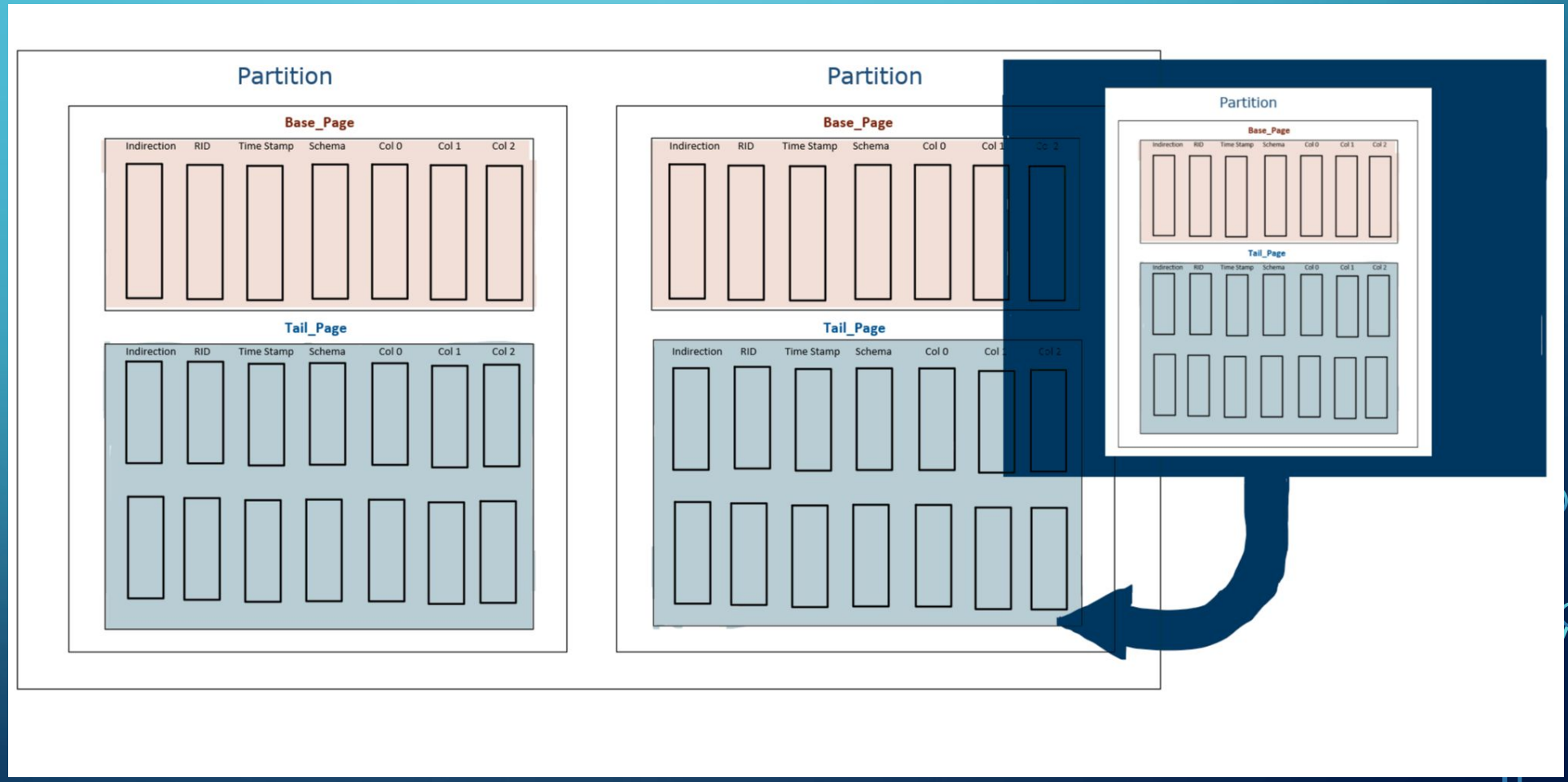
def index(self, value, n, first_only=TRUE)

QUERY INTERFACE

- simple query capabilities
  - Insert
  - Select
  - Update
  - Sum

QUERY INTERFACE

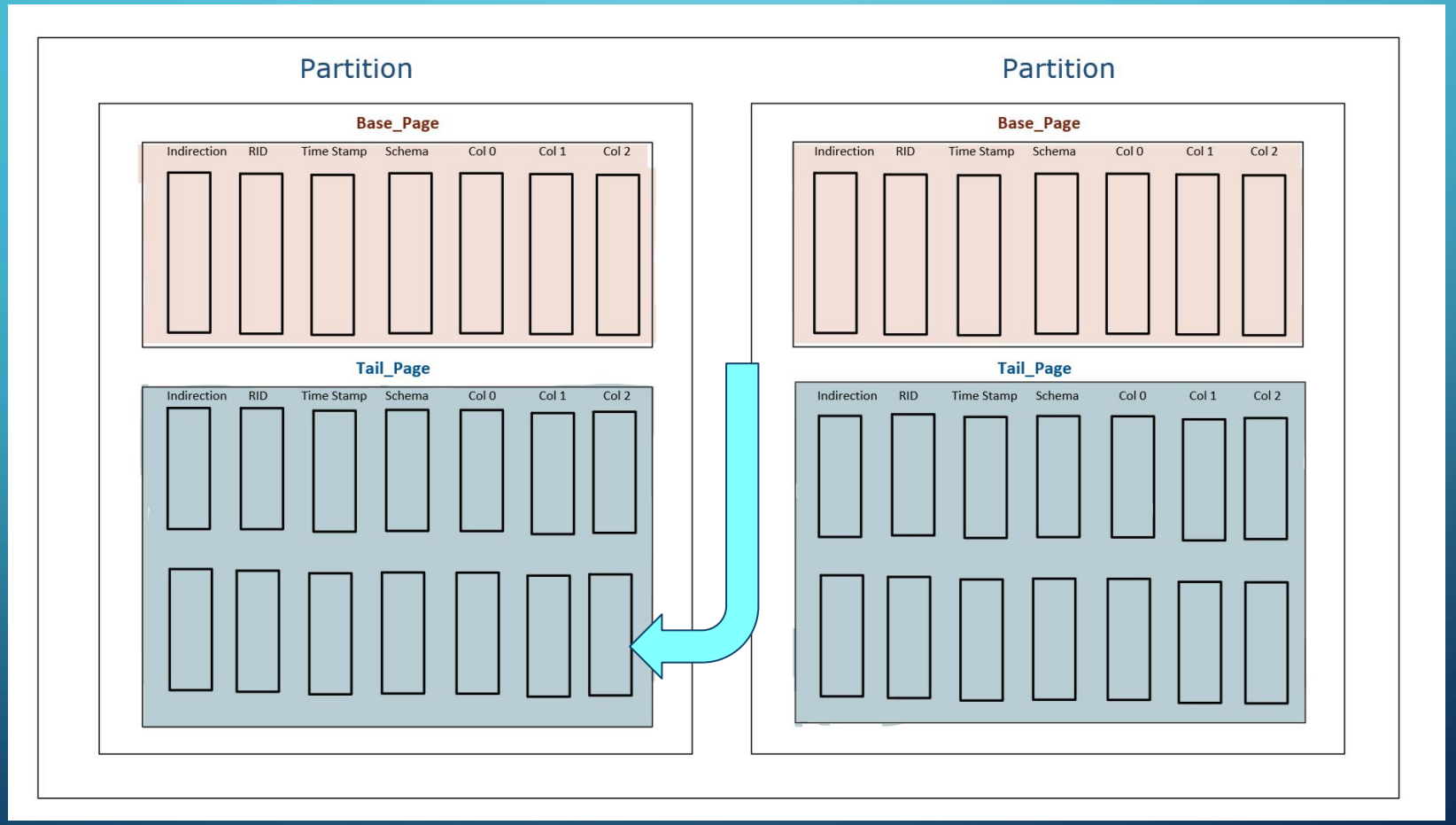
Insert



Query Interface --Select

- def select(self, key, query_columns)
- key: value of the primary key
- query_columns : list of boolean values

Query Interface --Select



Query Interface --Sum

- def sum(self, start_range, end_range, aggregate_column_index)

- Utilize the select() function in a single for loop

Query Interface --Update

- Cumulative
- For a base-record, there are mainly two-cases:
  - For both of the cases, we will both update the indirection to the latest record in tail page
  - Case 1: There has been at least one indirection
    - Follow the indirection and go to that record in the tail page
    - Combine the input columns with that record
    - Add it to the end of the tail page
  - Case 2: There's no indirection
    - Add the input columns to the end of the tail page

Query Interface --Update

- We trade off between update and read performance by creating our tail records to be either:

1. **Cumulative**

Better performance for Read()

2. **Non-Cumulative**

Better performance for Update()