# MEDB

## Milestone #1

Erik Linsenmayer

Nicholas Phomsopha

Tanner Eckmann

Trevor Hansen

Abdullah Mardini

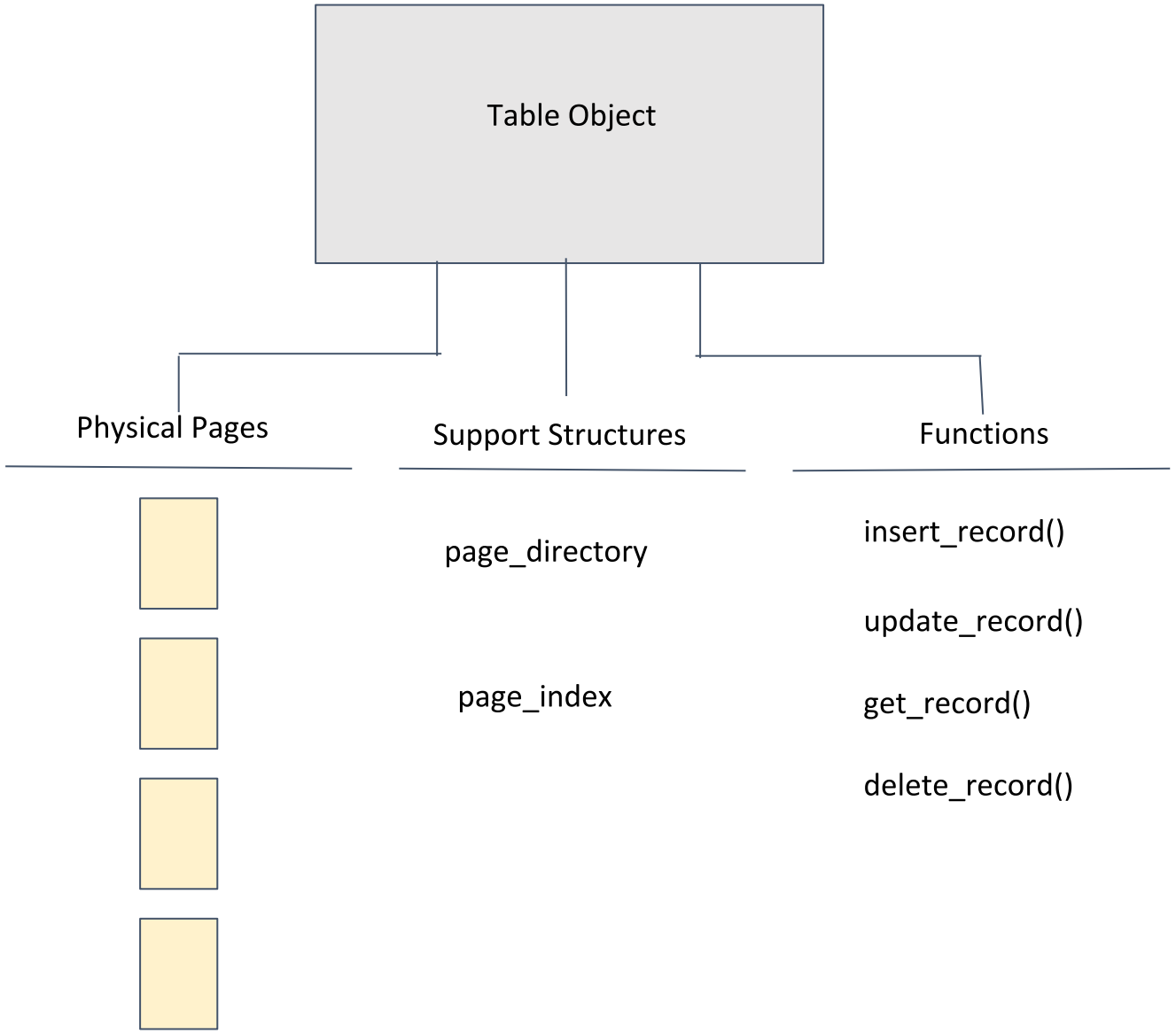
# The Table

Each table object maintains a list of all physical pages containing all data for all records (base and tail) associated with the table.

Query functions are essentially wrapper functions for the functions in the table class, where most of the work is done.

These table maintains a set of structures and variables that are used by the table functions to locate record-related data

The most important of these structures are the page directory and page index

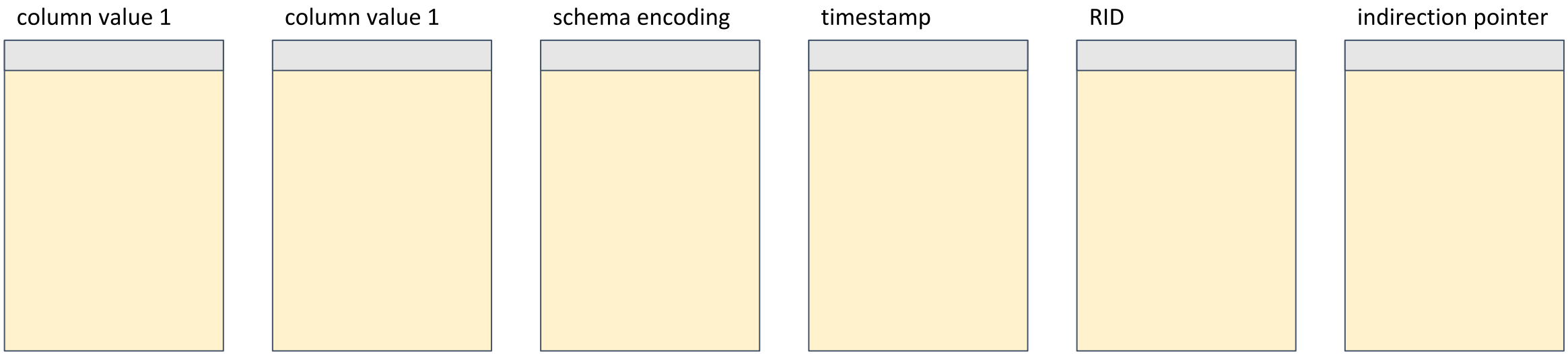


# Column-Oriented Storage

As required, data storage is column-oriented. Below is an example of how the data for a 2-column table with 1 record would be stored

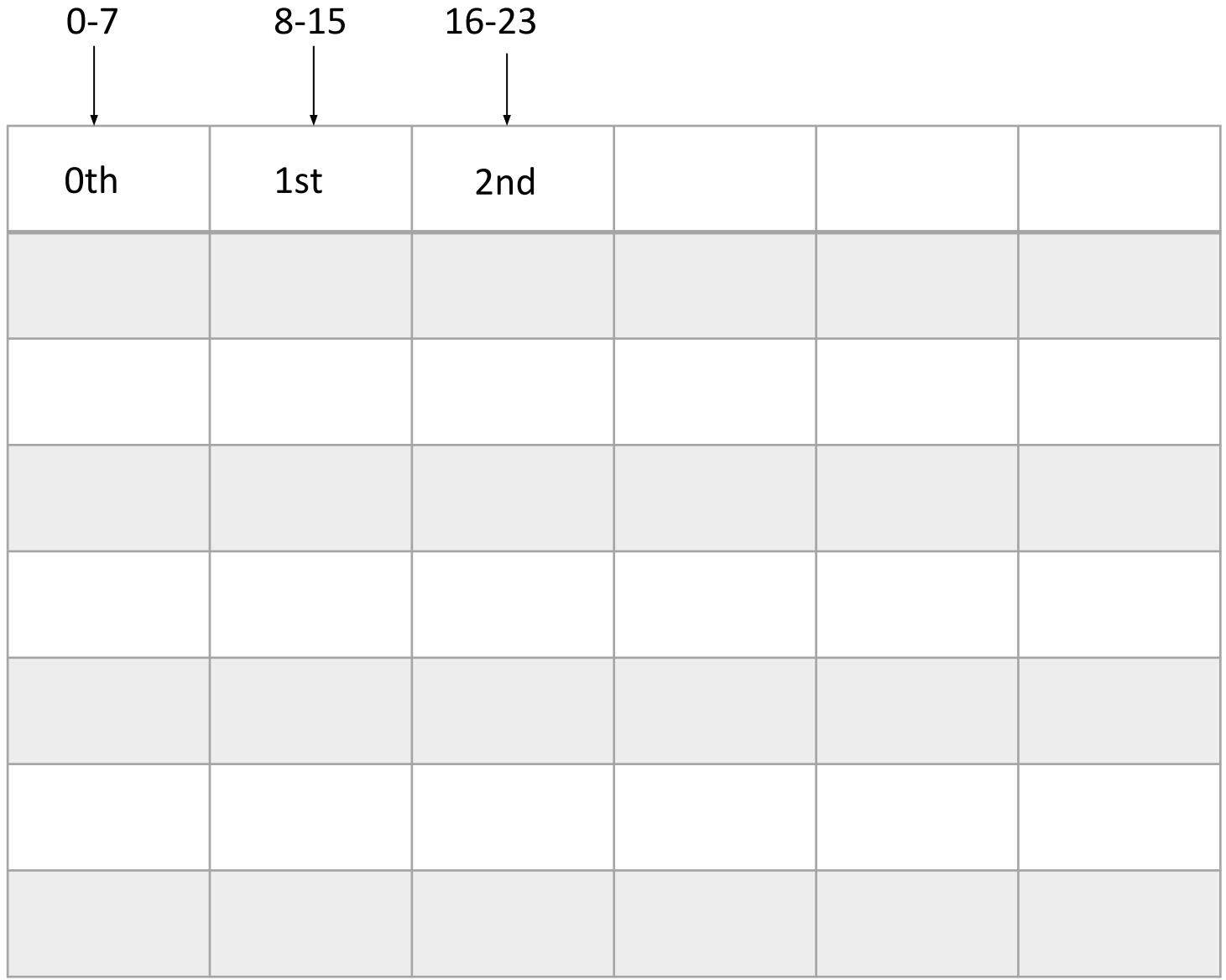
.

Any given record (base or otherwise) will span num_columns + 4 pages

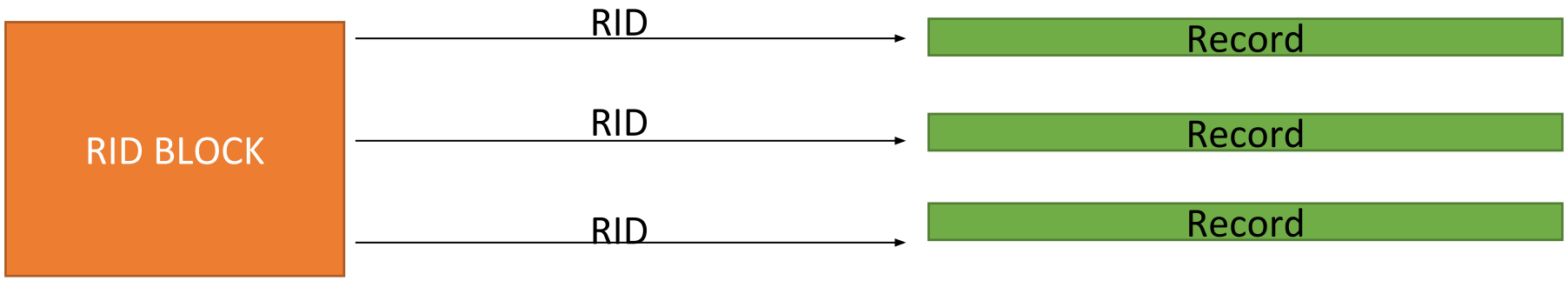


# Record Offset

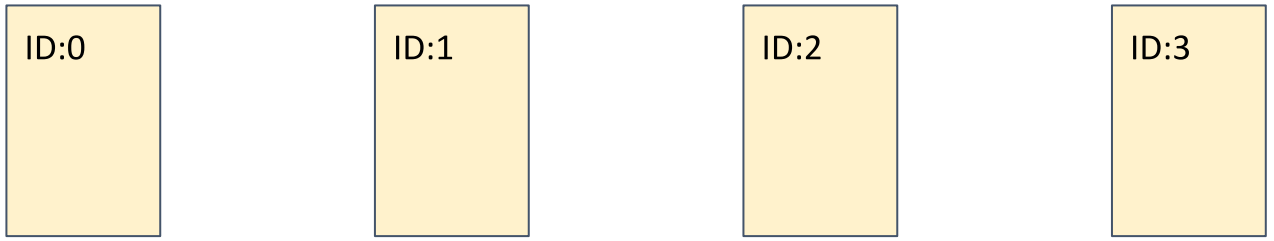
- Table maintains a Record offset.
- Record offset is used to fetch record data within a given page
- For example bytes 8-15 is the fist offset, 16-23 is the second and so on.



# RIDs and Page ID's

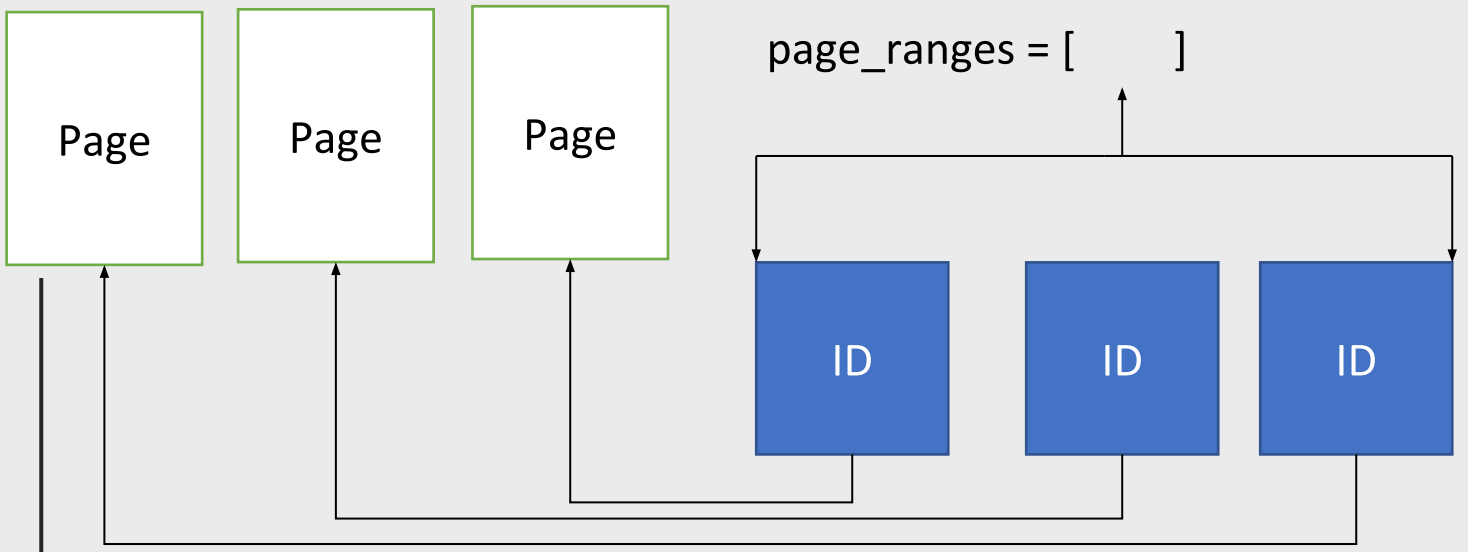



To ensure every record gets assigned a unique RID, each table object is given 'block' rids to pull from and give to inserted records. The table receives this block from a RID space allocator object. When the table runs out of RIDs, it requests more from the  allocater.



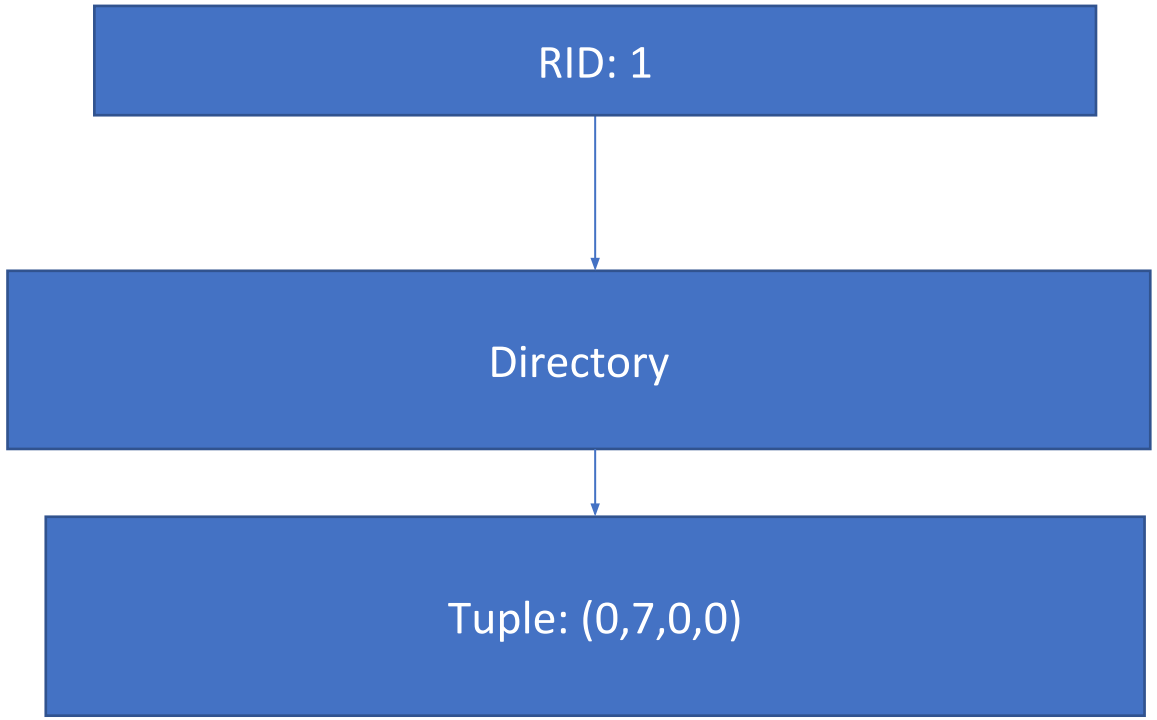

Every physical page is assigned a unique numerical ID (page id) by the table.

# The Page Range



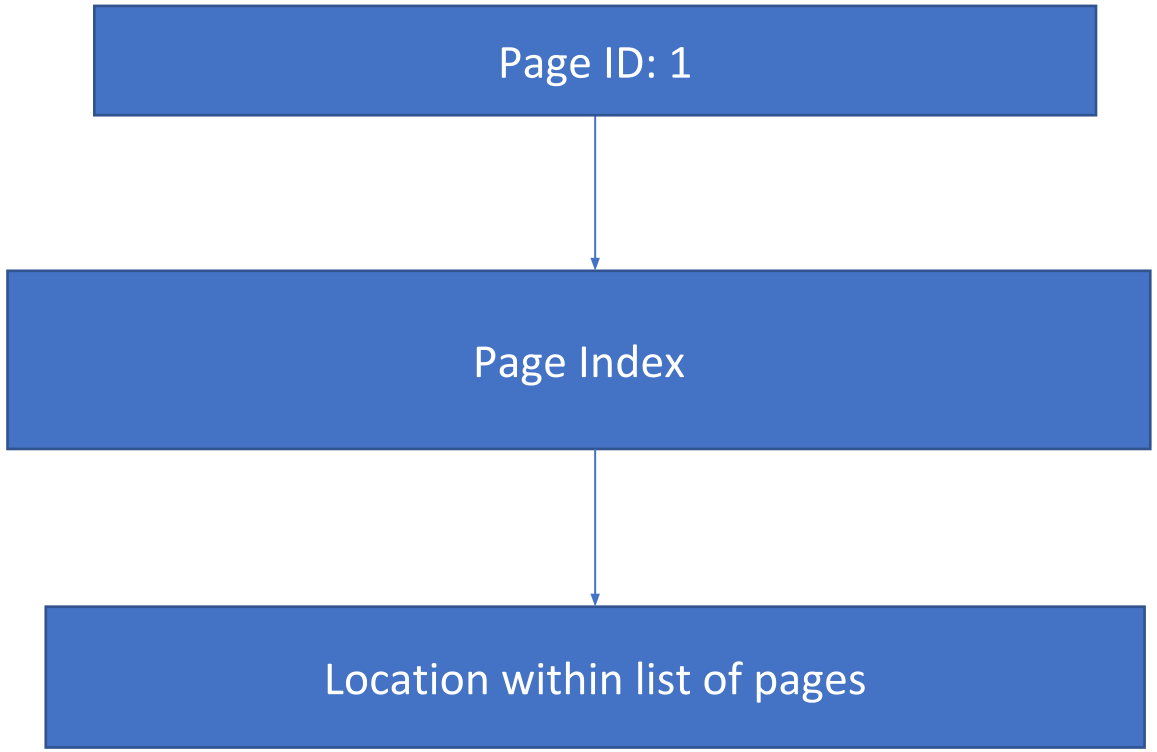

- The page_ranges only contain a list of page ids for the physical page in a page range. Each range has 512 records max.
- When it is filled up, a new set of base pages are allocated as well as another page range is made, this is only done if the previous base pages and page range are full.
- Example - we create a new table of four columns and insert 512 records into this table. The data for these records will span the first set of 4 base pages and 4 metadata base pages allocated for these records (say they have ids 1,2,3, and 4). Then page_ranges would look like: [[1,2,3,4,5,6,7,8] . Then we insert more records and we see the current page range is full.
    - We then allocate a new set of base pages and new page ranges, it will look like this [[1,2,3,4,5,6,7,8][9,10,11,12,13,14,17,18]]

# Page Directory




- The Page Directory maps each rid to a python tuple.
- The tuple contains the range of all page ids for all pages containing that record data.
- The record offset and the page range index is also present
- Implemented through a hash table

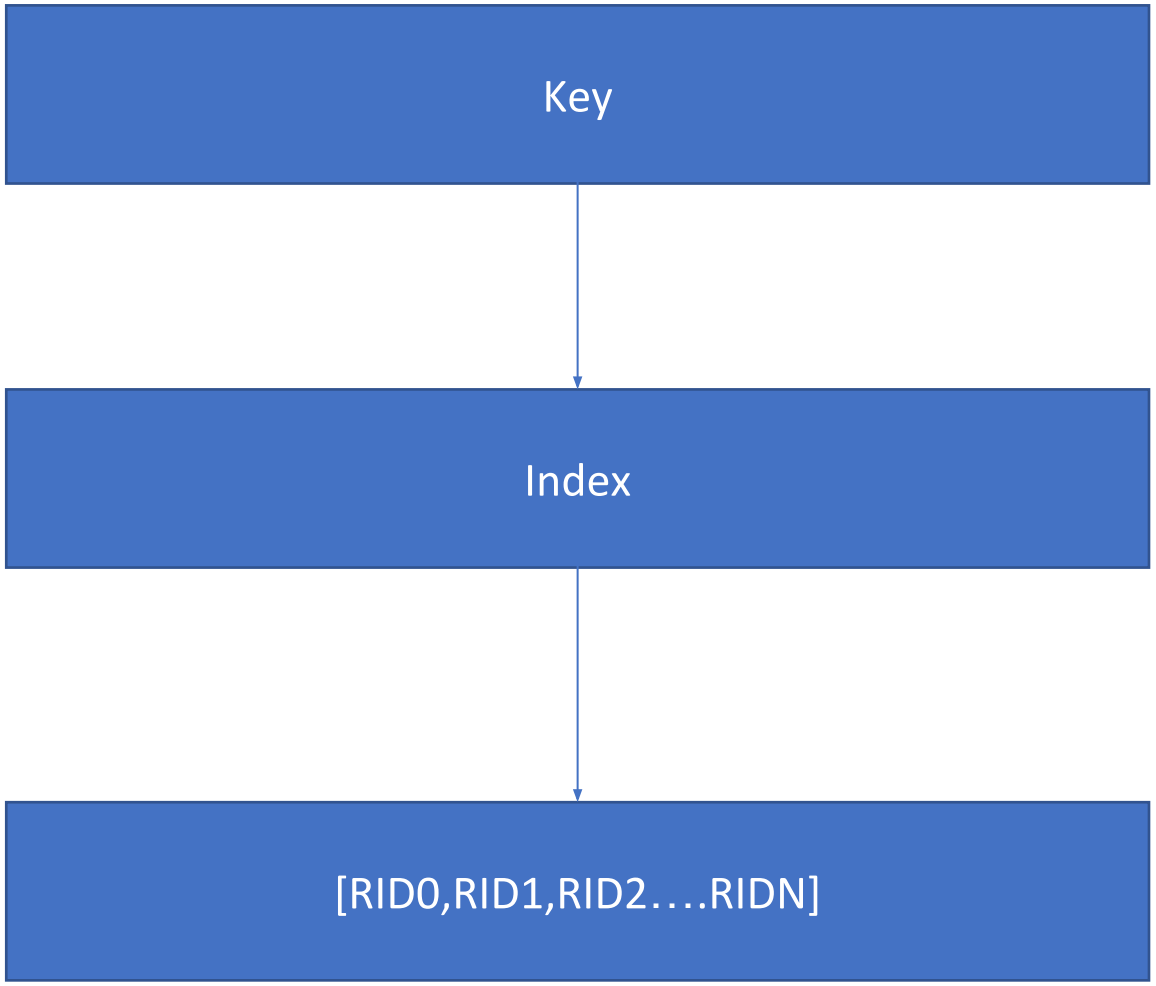
# Page Index




- The Page Index maps a page id to the location of the corresponding page in the pages list.
- Implemented through a hash table

# Index

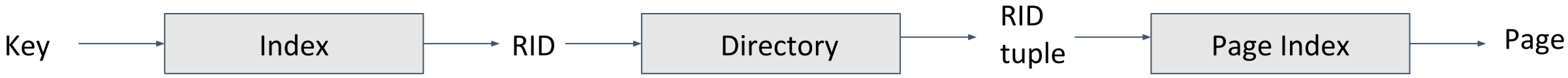
- The index maps a key value to a set of RIDs, where each RID corresponds to the record that contains the key value.
- Invisible to table. Used only by query class.
- Implemented as a hash table



# Obtaining pages for reads/writes

All functions will essentially follow same set of steps to access page data or to modify page data:

1. use key to get rid(s) via index (this occurs in the query object)
2. use rid to get ‘rid tuple’ via page directory
3. use the information (page id’s, offset, etc) contained in rid tuple to get data, passing page ids through the page index to obtain the desired page.
4. Read or write to page



# Insert

1. If set of base pages in most recently created page range is full:
   a. allocate fresh set of base pages.
   b. write column data and metadata to pages
   c. update page directory and page index
2. Otherwise:
   a. obtain set of base pages from most recently created page range
   b. write column data and metadata to these pages using offset
   c. update page directory and page index
3. Update index with key and rid pair

# Update

1. Allocate a set of tail pages to store updates if no tail pages have been allocated for the set of base page containing the updated record.
2. simply insert tail record data into these tail pages. Set the indirection pointer for this tail record to the indirection pointer for the updated base record
3. update indirection pointer for updated base record, so that it contains RID of newly created tail record.

# Get Record

1. using index, page directory, and page index, get pages containing requested record data.
2. check indirection column of record.
3. If indirection pointer is not null:
   a. obtain schema encoding of base record and go to tail record via the rid contained in base record's indirection column
   b. consult tail record's schema encoding, pull out appropriate data and go to next tail record (via current tail record's indirection pointer)
   c. continue until we reach end of lineage or until we've collected all of the most up to date data. (i.e until tail_schema0 | tail_schema1 | … = base_schema)
4. Otherwise, just read in requested data from base pages