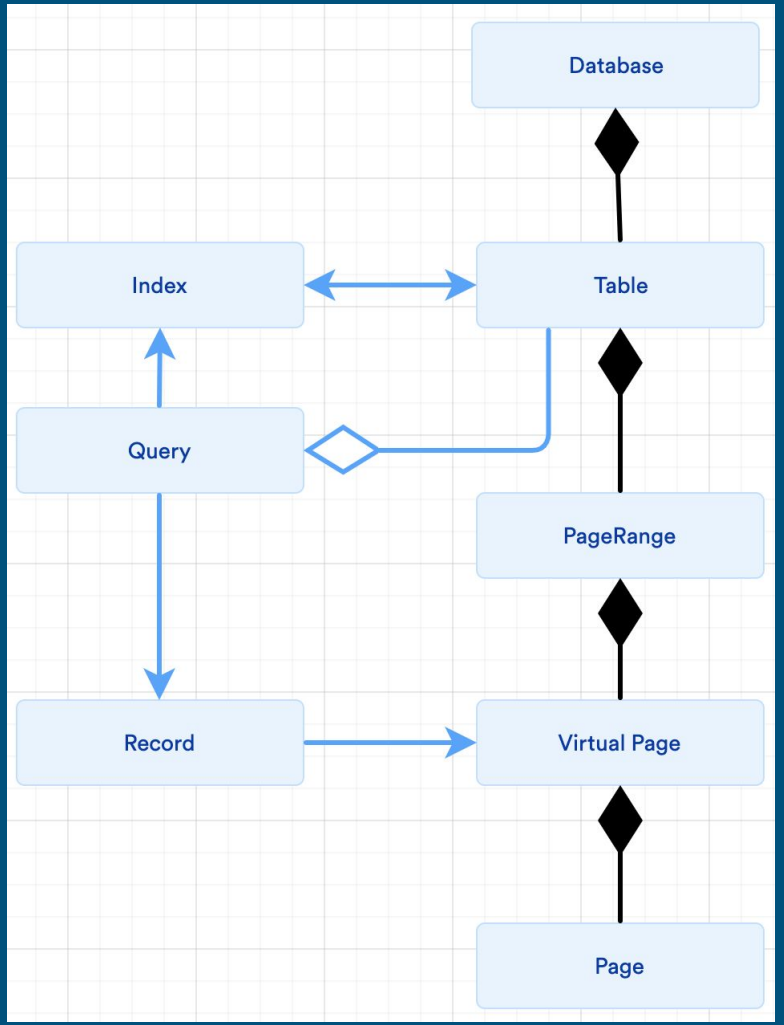
# Milestone 1


## Team SADCS

Soumya Duriseti, Alvin Agana, Daniel Silva, Christopher Pires, Shivani Parekh

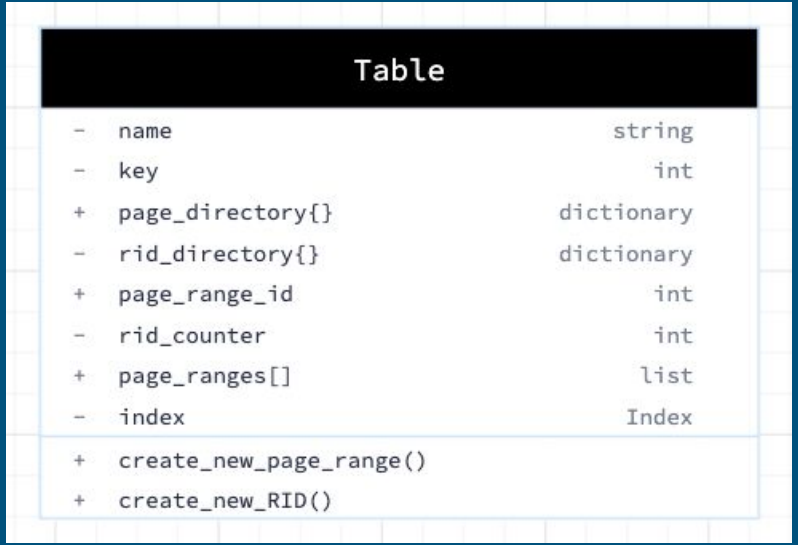
# Design - Overview

- Database -> dictionary of tables indexed by name
- Table -> list of PageRange objects
- PageRange -> list of VirtualPage objects
    - List of base pages and list of tail pages
- VirtualPage -> list of Page objects
    - Each Page stores a column of data from the Table
- Query has a Table to make queries to
- Query depends on Index and Record to perform some functions
- A Record is stored in a VirtualPage across Page objects



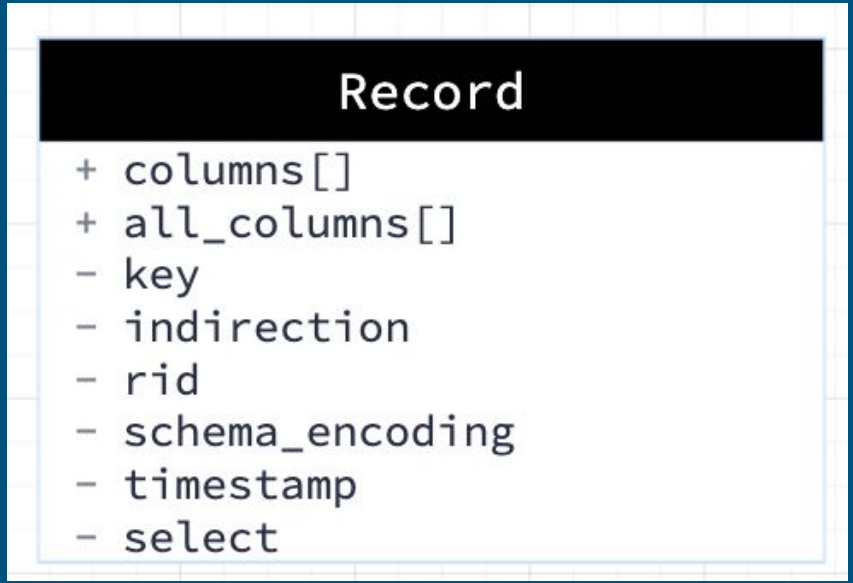
# Design - Table

- A table is created for organizing user specified data (eg. Grades)
- Creates and stores page ranges in a list
- Initializes an Index object
- Main components
  - RID_directory
    - Key: primary key
    - Value: RID
  - page_directory
    - Key: RID
    - Value: object {page range ID, row in page, virtual page id} (address of data)



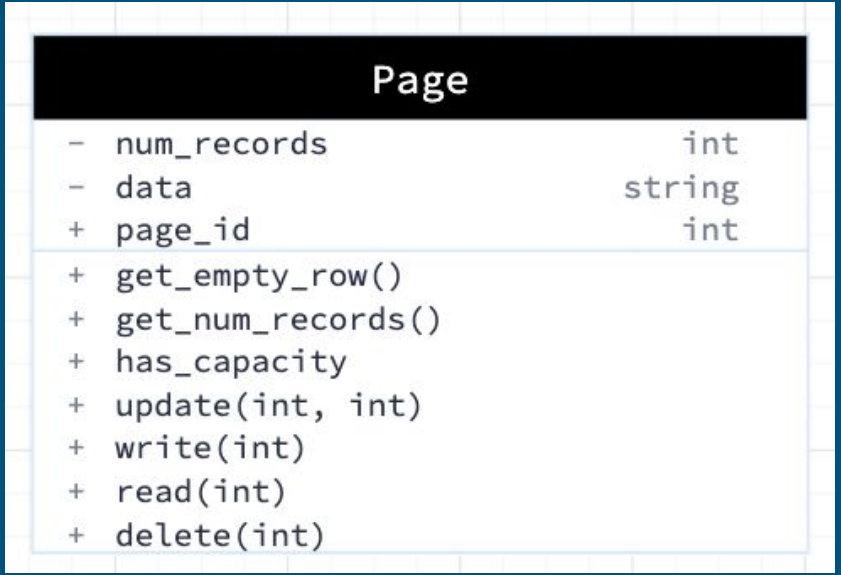
# Design - Record

- Created during insert() and update() only
- Initialized with 4 metadata columns
  - If being created by select, they are omitted
  - Appended to the first 4 columns of a record
  - Timestamp is noted upon record creation



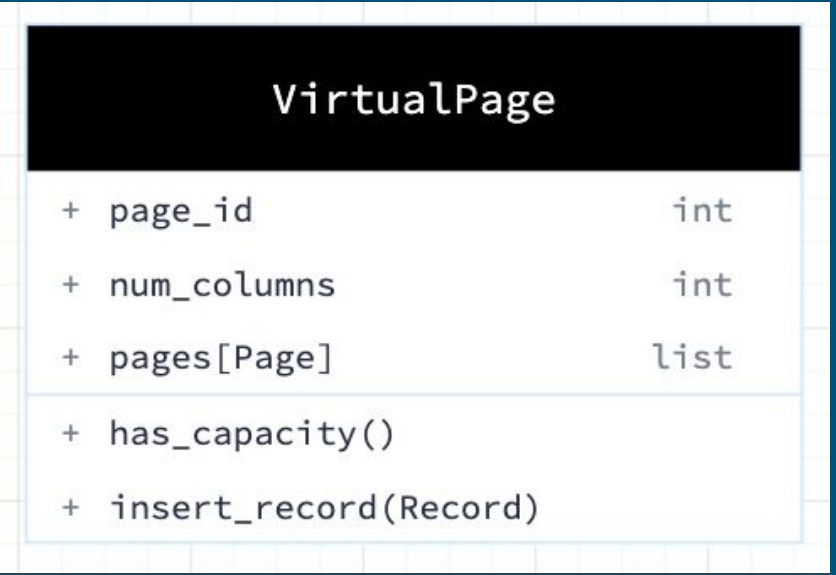
# Design - (Physical) Page

- Each page
  - Represents a column/attribute
  - Capacity: 4096 bytes (512 records)
- write()
  - Write to any 8 byte offsets within array
- delete()
  - Sets value to 0
- read()
  - Returns 8 bytes
- update()
  - Used to update indirection and schema encoding of base records



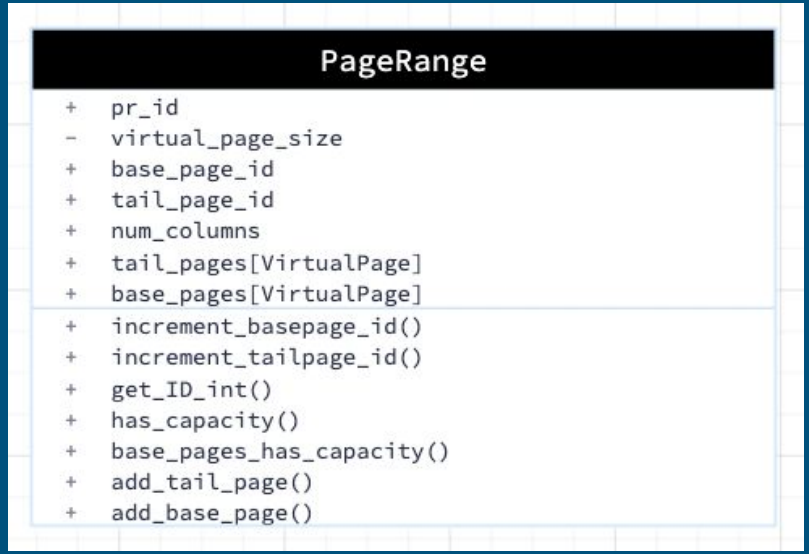
# Design - Virtual Page

- Parent class for basePage and tailPage, which will be further implemented in future milestones
- Manages physical pages as a list
  - Creates them
  - Stores them
  - Inserts records into them



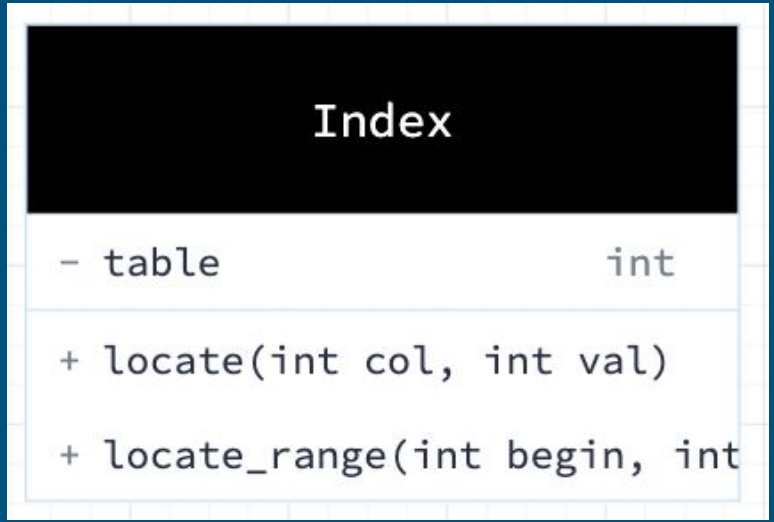
# Design - Page Range

- Each Page range:
  - Has a 512kb capacity (64k records)
    - Capacity checked using size and count of virtual pages
  - Holds 2 arrays: [base pages] & [tail pages]
- Creates and stores virtual pages
  - Virtual page ID's stored as strings (eg. "B_2", "T_1")



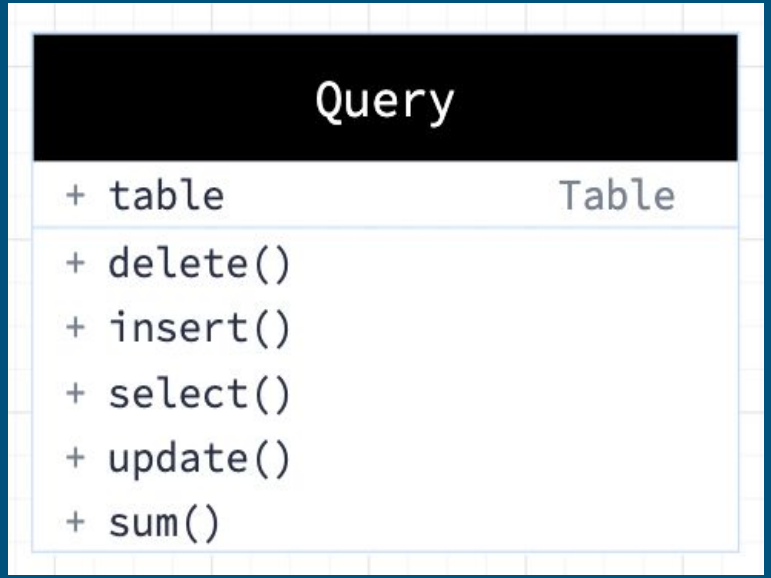
# Design - Index

- Used by select() in query to access read specific data
- locate()
  - Returns a list of RIDs
  - Looks at every base page RID, and iterates through tail pages when there has been an update to that record
- locate_range()
  - Same implementation as locate() except checks primary key of record is within user specified range



# Design - Query

- Query object created by user on a Table
- delete()
  - Gets address from rid_directory and page_directory
  - Invalidate the base record and all its tail records
    - Set RIDs to -1
- insert()
  - Creates new virtual page or page range if needed
  - Makes a new record object with new rid
  - Inserts into next available spot in base page
  - Updates page and rid directories in Table



# Design - Query

- select()
  - Calls locate() in Index class -> list of rids for matching records
  - Iterates matching records to return the most updated values from base  and tail pages
  - Returns data for columns requested in query columns
- update()
  - Non-cumulative updates
  - Inserts new tail record into tail page
    - Updates indirection of base page, tail page indirection points to previous update
- sum()
  - Given start and end of primary key range, and column to aggregate
  - Calls locate_range() in Index and receives a list of RIDs with primary keys in the range
  - Iterates list of rids to sum the most recent value for each record in aggregate column

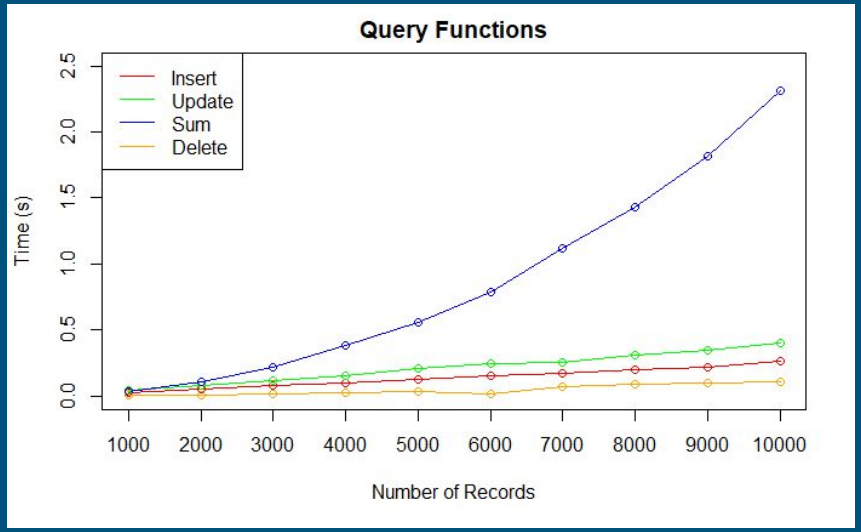
# Unit Tests

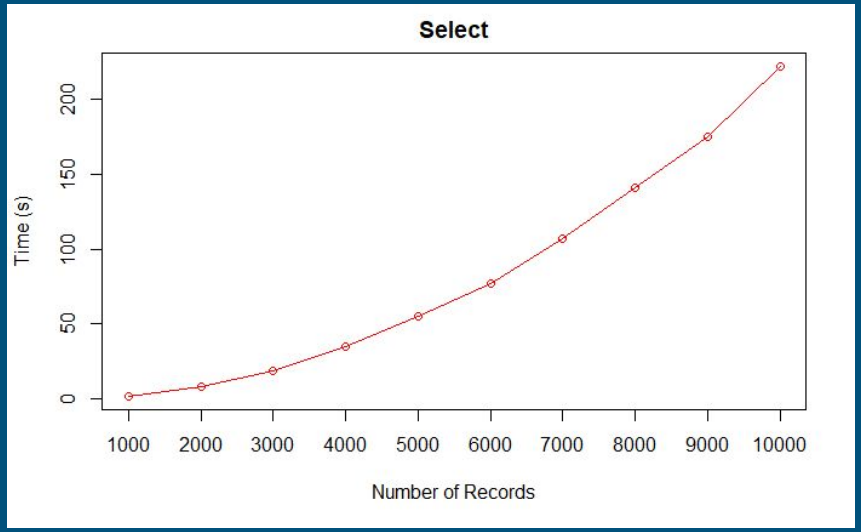
- Created expansive unit testing suite to test out all queries
  - Second version of test suite will feature different amount of records and table sizes for each operation tested

```
def test_insert_one(self):
    # arrange
    tbl = self.makeSampleTable("Grades", 5, 0)
    query = Query(tbl)
    # act
    query.insert(69420, 0, 0, 0, 0)
    # assert
    self.assertAlmostEqual(tbl.page_ranges[0].base_pages[0].pages[4].read(0), 69420)
```

# Results

| | |
|---|---|
| Processor | 2.3 GHz Dual-Core Intel Core i5 |
| Memory | 8 GB 2133 MHz LPDDR3 |
| Startup Disk | Macintosh HD |
| Graphics | Intel Iris Plus Graphics 640 1536 MB |



# Challenges

During the initial stages we struggled understanding the logical view of the database and meaning of terms

- Why we needed page ranges
- How base and tail pages were represented
- What data structures to use
- Getting accustomed to python as coding language

# Future Improvements

- More helper functions and methods to clean up code
- Use a data structure like a tree for indexing
  - A more efficient select function
- Use cumulative updates for faster reads
- Expanding unit testing suite to include all functions and classes