L-Store
Milestone 1

ECS 165A
Natheenthorn Teachaurangchit
Michael Shaw
Stuart Feng
Henry Chou
Eric Wang
Data Model
- Data Storage
- Base and Tail Page
- Page Range

Bufferpool Management
- Page Directory
- Index Directory

Query Interface
- Select
- Update
- Sum
- Insert
- Delete
- Increment
(S1) Data Model
Column-based Data Storage

Metadata (Indirection, RID, Timestamp, schema encoding)

Data Values

Page 1

Page # = num_columns + METADATA_CT (4)
Design decision (Python Objects)

Initially (All the pages were stored in an array):

```python
[
    [[ base_page ], [ base_page ]], #Page Range 1
    [[ base_page ], [ base_page ]], #Page Range 2
]
```

Final Decision (Store everything in an object):

```python
[ PageRange( BasePage(), ... ) ]
```

**Page Size** = 4096 bytes, **Record Size** = 8 bytes

Therefore, we could store 512 records per page
Each page range should be able to store 8192 records in the base pages. This should make it easier to ease into the next milestones of dividing the workload across multiple threads without concurrency issues.
(S2) Bufferpool Management
Decision (HashMap): Since all the RIDs are unique, there should be no collision when hashing; therefore, HashMap would give us the best performance.
Index Directory

Design Decision (B-Tree): Since we are implementing index for all of the columns, we acknowledge that there will be columns with duplicate values. Therefore, using B-Tree will yield the best performance, while maintaining appropriate memory allocation.
(S3) Query Interface
Select

```python
select(self, index_value, index_column, query_columns)
```
Insert

```
insert(self, *columns)
```

Base Page

New Inserted Records

Tail Page
Update

update(self, primary_key, *columns)
Sum

$\text{sum}(\text{self, start\_range, end\_range, aggregate\_column\_index})$

Straight forward check for range of columns and aggregates the sum of their values

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Page Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID (xxx-xxx-xxxx)</td>
<td>(Page Range, Page ID, Slot Number)</td>
<td></td>
</tr>
<tr>
<td>RID (xxx-xxx-xxxx)</td>
<td>(Page Range, Page ID, Slot Number)</td>
<td></td>
</tr>
<tr>
<td>RID (xxx-xxx-xxxx)</td>
<td>(Page Range, Page ID, Slot Number)</td>
<td></td>
</tr>
<tr>
<td>RID (xxx-xxx-xxxx)</td>
<td>(Page Range, Page ID, Slot Number)</td>
<td></td>
</tr>
<tr>
<td>RID (xxx-xxx-xxxx)</td>
<td>(Page Range, Page ID, Slot Number)</td>
<td></td>
</tr>
</tbody>
</table>

Global SUM = 0
Delete

delete(self, primary_key)

Straight forward deletion method using provided primary key to use equivalent RID and remove target record in base page.
Roadmap

- Implemented query interface functions, without page range.
- Index was replaced with a B-Tree implementation.
- Page Range implemented and integrated.
- First functional implementation, problem with insert function being very slow.
- Alternative implementation proposed and worked on. Problems arose query functions being much slower.
- Alternative implementation problems fixed (index was not being used), is now our main implementation.
Current & Alternate Implementation Speed Comparison

Run Times (s)

- Current Implementation
- Previous Implementation

- Insert
- Update
- Select
- Sum
- Delete

The current implementation has significantly higher run times compared to the previous implementation for all operations.
Thank you!