Objectives

- Data Model
- Bufferpool
- Query API
- Performance
- Questions
- Demo
Lineage Based Data Store
A relational columnar database designed to bridge the gap between OLTP and OLAP workloads.
L-Store Database

Table 1
- Page Directory
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records

Table 2

Table 3

Table n-1
- Add as needed

Table 1
- Page Directory
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records
- Page Range 1 - 8 pages, records

Page Range 1
- Base Page 1
- Base Page 2
- Base Page 3
- Base Page 4
- Base Page 5
- Base Page 6
- Base Page 7
- Base Page 8

Base Page 1
- records 0 - 511

Physical Page
- 512 entries
- 8 bytes per entry
- 4096 kB total

Meta Data
- indirection
- rid
- schema encoding
- timestamp
- primary keys
- column 1 data
- column 2 data

Tail Pages
- append only record

read only record
entry
### Base Page
- Read only
- Holds 512 records

### Tail Page
#### Directory
- Appends only records, no upper limit on records

#### Tail Pages

<table>
<thead>
<tr>
<th>Indirection</th>
<th>rid</th>
<th>rid encoding</th>
<th>timestamp</th>
<th>primary keys</th>
<th>column 1 data</th>
<th>column 2 data</th>
</tr>
</thead>
<tbody>
<tr>
<td>509</td>
<td>502</td>
<td>3:30</td>
<td>6688</td>
<td>99</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirection</th>
<th>tid</th>
<th>schema encoding</th>
<th>timestamp</th>
<th>primary keys</th>
<th>column 1 data</th>
<th>column 2 data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>509</td>
<td>2</td>
<td>5:30</td>
<td>MAX_INT</td>
<td>100</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirection</th>
<th>tid</th>
<th>schema encoding</th>
<th>timestamp</th>
<th>primary keys</th>
<th>column 1 data</th>
<th>column 2 data</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.00</td>
<td>509</td>
<td>2</td>
<td>5:30</td>
<td>MAX_INT</td>
<td>100</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>
Physical Page Format

Finding an entry in the physical page:

- Physical page = bytearray of 4096 bytes, each column entry taking 8 bytes. This makes for 512 entries per physical page.
  - Bytearray size and record size can be configured as constants
- **Starting byte index:** row * PAGE_RECORD_SIZE
- **Ending byte index:** starting point + PAGE_RECORD_SIZE
- Writes positive integers only. Conversion into other data types (timestamp, None, string) is handled at the table level.
  - Write converts integer into 8 bytes
  - Read converts bytes back into integers
  - Special null value: $(2^{64})-1$
Bufferpool

Page Directory maps RIDs to their corresponding page range, base page, and physical page index. RIDs are assigned using record count.

Page.Directory = { 64803: { "page_range": 7, "base_page": 14, "page_index": 291}}

def __rid_to_page_location(self, rid: int) -> dict:
    ...  
    Helper function that returns a dict of the memory location for a given RID
    ...
    page_range_index = math.floor(rid / ENTRIES_PER_PAGE_RANGE)
    index = rid % ENTRIES_PER_PAGE_RANGE
    base_page_index = math.floor(index / ENTRIES_PER_PAGE)
    physical_page_index = index % ENTRIES_PER_PAGE
    return { 'page_range': page_range_index, 'base_page': base_page_index, 'page_index': physical_page_index }
<table>
<thead>
<tr>
<th>Query API</th>
<th>Insert</th>
<th>Update</th>
<th>Select</th>
<th>Delete</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert</strong></td>
<td>1. Validate user args &lt;br&gt; 2. Make new RID based on record count, set schema to 0, create new record object &lt;br&gt; 3. Add RID to page directory and write the record</td>
<td>1. Check record exists and validate user args &lt;br&gt; 2. Get the most updated record &lt;br&gt; 3. Update schema encoding and add new updated values to columns &lt;br&gt; 4. Create a new TID and record, set indirection &lt;br&gt; 5. Write to tail page, change indirection and schema of base page</td>
<td>1. Check record exists and validate user args</td>
<td>1. Check record exists based on the key</td>
<td>1. Loop through all keys and check if the key is between the start range and end range</td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>1. Check record exists and validate user args</td>
<td>2. Search for the key based on given column index and return the RID for any matches</td>
<td>2. If exists, return the RID for that record</td>
<td>2. If between ranges, add the value at the aggregate column index to the running sum</td>
<td>2. If between ranges, add the value at the aggregate column index to the running sum</td>
</tr>
<tr>
<td><strong>Select</strong></td>
<td>1. Check record exists and validate user args</td>
<td>3. Get the most recent updated record using RID</td>
<td>3. Set the “Deleted” value in our page directory to True</td>
<td>3. Set the “Deleted” value in our page directory to True</td>
<td>3. Return the sum or False if no records fall between the given range</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>1. Check record exists based on the key</td>
<td>4. Future calls to this record will return False for this key</td>
<td>4. Future calls to this record will return False for this key</td>
<td>4. Future calls to this record will return False for this key</td>
<td>4. Future calls to this record will return False for this key</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td>1. Loop through all keys and check if the key is between the start range and end range</td>
<td>5. Return a list of records</td>
<td>5. Return a list of records</td>
<td>5. Return a list of records</td>
<td>5. Return a list of records</td>
</tr>
</tbody>
</table>
Optimizations & Performance

- RID to page location is $O(1)$
- Validate user arguments before writing a record
- Integers for schema encoding

*These times are based on 10 run averages using the provided `__main__.py`

Processor: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, 4001 Mhz, 4 Core(s), 8 Logical Processor(s)
Questions