L-Store Milestone 2

Alana Rufer, Eseosa Omorogieva, Nina Gopal, Riddhi Barbhaiya, Kushaal Rao
Review of logical memory organization

RID → [page_range_offset, page_index, offset]
Indexing
Index Structure

Array of dictionaries, with the index of each dictionary corresponding to the column number.

Columns without indexes have a None placeholder.

Value are mapped to a list of RIDs of records that contain the value in that column.
**Index Create and Drop Column**

1. Fetch all the Page IDs that are a part of the column.
3. Fetch all the Page IDs that contain a field for a record.
4. Check if the record is in the column.
5. If the record is in the column, we fetch the field from the bufferpool.
6. Insert the RID and value to the Index.
Durability
File Organization & Structure

- 3 files
- Separate files for data and for table & bufferpool information
- `db.open()`: restore from files
- `db.close()`: flush dirty pages to data file and store other information in objects file
# Physical Page Metadata

Moved metadata inside the 4096 block allocated to a physical page

Added **TPS** (Tail Page Sequence Number): RID of last merged tail record

TPS is used to help determine whether a field is up to date

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_records: 2 bytes</td>
</tr>
<tr>
<td>TPS (Tail Page Sequence #): 8 bytes</td>
</tr>
<tr>
<td>slot 0 : 8 bytes</td>
</tr>
<tr>
<td>slot 1</td>
</tr>
<tr>
<td>slot 2</td>
</tr>
<tr>
<td>slot 3</td>
</tr>
<tr>
<td>slot 4</td>
</tr>
<tr>
<td>slot 5</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>slot 507</td>
</tr>
<tr>
<td>slot 508</td>
</tr>
<tr>
<td>slot 509</td>
</tr>
</tbody>
</table>
Bufferpool

Abstracts disk accesses

Shared by all database tables

Granularity of physical pages

Uses locks to prevent conflict between merge and user threads

Deallocates disk space when a table is dropped
Bufferpool

**pool_index**: index of physical page in bufferpool and pins array

**pageid**: virtual page identifier

**frameid**: disk location identifier

---

**Bufferpool**

**pool** [size = BUFFERPOOL_SIZE]

```
| Page | Page | Page | Page | ... | Page | Page |
```

**pins**

```
| count | count | count | count | ... | count | count |
```

---

**frame_map**

- **frameid**: disk location identifier

**pool_map**

- [**pool_index**, **dirty_bit**]

---

**lru_manager**

- **least recent**
  - pageid
  - pageid
  - pageid
  - pageid
  - ...
  - pageid

- **most recent**
  - pageid
  - pageid

---

**Data file**

- fetch page
- write page
Page Replacement (LRU)

request page by pageid

page in pool?

space in pool?

no

no

evict LRU page

fetch page from disk into pool

yes

return page

Evict page from bufferpool

least recent

lru_manager

pageid

pageid

...)

pinned?

yes

no

dirty?

frame_map

pool_map

[remove pageid]

data file

Fetch page from disk

requested pageid

frame_map

data file

pool

... [open]...
Merging
Initiating Merge

Merge after reaching threshold value of filled tail pages
Copy To Merge Pool

Separate storage area for pages used in the merge

Individual Page ID

Page in pool_map?

no

Get from Disk

yes

Get from Pool

Physical Page

Copy of physical page

Merge Pool
Merging

Loop through tail records

Logical Tail Page

Base rid | Tail rid | ... |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: if base_rid is seen, skip

For each tail record

Check schema

Find physical base page IDs of columns to update

In merge pool?

no

Copy To Merge Pool

Loaded physical base pages

yes

Update physical base page with the tail fields
Concluding Merge

Bufferpool

Disk (data file)

{ Physical base pages with updated values }

Page Range

Logical page

New PageID New PageID ... New PageID

Page Range
Performance
Tuning Bufferpool and the Threshold for Merging

Finalized Merge Threshold = 10
Finalized Bufferpool = 200

Workload: tuning_merge.py, tuning_buffer.py
Hardware: Dual-Core Intel Core i7, 2.5GHz, 16GB, 4 MB L3 Cache
Query performance

Workload: `__main__.py`

Hardware: Dual-Core Intel Core i7, 2.5GHz, 16GB, 4 MB L3 Cache
Q&A