DB storage architectures:
Rows, Columns, LSM trees

COS 518: Advanced Computer Systems
Lecture 7
Michael Freedman

Basic row-based storage

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>CHAR(32)</td>
<td>INT</td>
<td>SMALLINT</td>
<td>DATE</td>
</tr>
</tbody>
</table>

Basic row-based storage

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>CHAR(32)</td>
<td>INT</td>
<td>SMALLINT</td>
<td>DATE</td>
</tr>
</tbody>
</table>

Basic row-based storage

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>CHAR(32)</td>
<td>INT</td>
<td>SMALLINT</td>
<td>DATE</td>
</tr>
</tbody>
</table>

Basic row-based storage

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>CHAR(32)</td>
<td>INT</td>
<td>SMALLINT</td>
<td>DATE</td>
</tr>
</tbody>
</table>

READ 32 bytes at positions (0 + 8), (50 + 8), (100 + 8), (150 + 8)
### Row-based storage: variable lengths

<table>
<thead>
<tr>
<th>8</th>
<th>0 - 255</th>
<th>4</th>
<th>2</th>
<th>4</th>
<th>= 18 - 273</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>URL</td>
<td>Size</td>
<td>Code</td>
<td>Fetched</td>
<td>DATE</td>
</tr>
</tbody>
</table>

**Id** | **BIGINT**
| **URL** | **VARCHAR(255)**
| **Size** | **INT**
| **Code** | **SMALLINT**
| **Fetched** | **DATE**

How do you walk through all the URLs? No longer at fixed offsets

### Row-based disk layout

- Data stored in fixed-sized pages on disk
  - E.g., typically 8K in PostgreSQL
  - Page includes metadata and actual data items
  - Items = indexes, data rows

### Row-based storage: variable lengths

**ItemData**: [(0, 18), (18, 273), (291, 59)]

<table>
<thead>
<tr>
<th>Id</th>
<th>URL</th>
<th>Size</th>
<th>Code</th>
<th>Fetched</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://www.postgresql.org/docs/9.5/static/storage-page-layout.html

### Row-based disk layout

- Data stored in fixed-sized pages on disk
  - E.g., typically 8K in PostgreSQL
  - Page includes metadata and actual data items
  - Items = indexes, data rows

**READ 32 bytes at positions (0 + 8), (50 + 8), (100 + 8), (150 + 8)**
Types of database workloads

- OLTP = OnLine Transaction Processing
  - Write-heavy
  - Transactions

- OLAP = OnLine Analytical Processing
  - Read-heavy
  - Analytical scans or “rollups” along column
    - “SELECT AVG(latency) FROM system WHERE time > now() – interval(“1h”)”

Comparison of disk layouts

- Row-oriented layout

- Column-oriented layout
  - Particularly good for compression, especially for long runs of identical numbers or small deltas

Good discussion of benefits of columns...

C-Store: A Column-oriented DBMS


Column-Stores vs. Row-Stores: How Different Are They Really?


LSM Trees: Discussion

- SSTable: set of arbitrary, sorted key-value pairs

- LSM Trees: Write to memory, then flush to disk
LSM Trees: Discussion

1. On-disk SSTable indexes are always loaded into memory
2. All writes go directly to the MemTable index
3. Reads check the MemTable first and then the SSTable indexes
4. Periodically, the MemTable is flushed to disk as an SSTable
5. Periodically, on-disk SSTables are "collapsed together"

• LSM Trees: Write to memory, then flush to disk