B-Trees, Fractal Trees, Heaps and Log Structured Merge Trees, Where did they all come from and Why?
AGENDA

Me and My Employer
Some Terminology
A quick Summary
The Heap
The B-Tree
Log structured Merge Trees
Fractal Trees
Column Stores
Conclusions
PRELIMINARIES
My database background in bullets

Punched Cards, Unit Record Machines - Univac 1004

Raw Disk – IBM 1620 and PDP-8

Random, Sequential and ISAM files

Hierarchical Databases – IBM IMS

Codasyl Network Database – Univac DMS 1100

Relational

  Tandem Enform (pre SQL)

  Relational – Tandem SQL/MP and MX

MySQL, PostgreSQL, Informix, Sybase

NoSQL

  Cassandra, Hbase, MongoDB, CouchDB, CouchBase
ABOUT PYTHIAN

Pythian’s 400+ IT professionals help companies adopt and manage disruptive technologies to better compete
EXPERIENCED

11,800 Systems currently managed by Pythian

GLOBAL

400 Pythian experts in 35 countries

EXPERTS

2 Millennia of experience gathered and shared over 19 years
Definitions

Latency – The time it takes to complete an operation A.K.A. response time

Write amplification – How many writes to storage it takes to store and keep a piece of data stored

Fragmentation – Unusable space that accrues in a database over time

Concurrency – The number of operations that can be performed on a database at the same time
Overview
Overview

Heaps – Old tech, fast to write initially, fragments badly with updates good for very low update high read rates. Especially with large numbers of indexes

B-Trees – MainStay of databases for some time, manages fragmentation up to a point, good for applications with lots of updates generally transactional, Excellent concurrency

Log Structured Merge trees – Very fast writes, doesn’t fragment, High write amplification for long lived data. Good for high write volumes with low to moderate read rates. Read latency is lower than write latency
Overview

Fractal trees – Merges features from B-trees with Log structured Merge trees. Good concurrency, fast writes slower reads and updates, Minimal fragmentation issues over time.

Column Store – Stores data by column not row, needs compression to save space, bad for concurrency, bad for reading whole rows. Very good for BI and Data ware house applications.
The Data Structures
**HEAP**

MySQL MyISAM and MongoDB MMapV1

Primary key not required

Data is initially written sequentially to the top of the heap

As rows are updated or deleted holes are left behind

The database engine will try to fill those holes

Unless all rows are exactly the same size to fill the holes small holes stay and are hard to fill. Fragmentation can become a problem
Heap

Indexes (which are themselves usually B-Trees) point to each row on the heap.

When a row moves all indexes for the table must be updated

All indexes are more or less equal in performance for retrieving rows

Writes, Updates and Deletes are much more expensive than Reads

Difficult to manage write concurrency

Good for low write rate high read rate applications – Catalog data (descriptions not inventory)
Heap
B-Tree

MySQL InnoDB and MongoDB Wild Tiger use B-Trees to store data

Must have a primary key

Data is stored inside the primary key B-tree at the leaf level

Fragmentation can occur but tends to be self limiting to 50%

Secondary indexes include the primary key

Primary key access is always quicker than secondary key access

Unless you are lucky to get index only scans

Writes, Updates and Deletes are much more expensive than Reads

B-Trees generally support good read and write concurrency

Very good for transactional work loads

Keeping track of Account balances Order processing etc
B-Tree
Log Structured Merge Tree

MyRocks, MongoDB RocksDB, Cassandra, CouchBase, Hbase, LevelDB

Writes, Updates and Deletes are treated the same

Data is written in “logs” with associated index tree(s)

Completed logs are never updated – eventually replaced

Lots of difference in implementation

Periodically the “logs” are merged together to compact out old data

Tend to have high write amplification for long lived data

Writes are an order of magnitude cheaper than reads

Very good for heavy write with light or moderate reads

Very good for data that expires

Time series, customer service logs, IOT data
Log Structured Merge Tree
Fractal Tree

TokuDB, TokuKV, TokuMX

Kind of a marriage between B-Tree and Log Structured Merge Tree

Non leaf level index blocks contain both index and row data as it inserted starting with the root node

As changes are made to the database, they start at the root node and migrate down to the leaf nodes passing through other level nodes as they go. These changes are referred to as messages

Latency

- Writes are very fast
- Key reads are almost as fast as B-Tree
- Sequential reads suffer from having to collect messages from the tree

Fragmentation is handled similarly to B-Tree

Much larger blocks than typical for B-Trees 2 megabytes typical
Fractal Tree
Column store

Vertica, MariaDB Column Store, ClickHouse

Tables must have primary key

Data is stored by column not by row

Writing is best done in large batches – Think ETL

Reading rows is possible but not recommended – its expensive

Column aggregates are inexpensive and quick

Very good for Business Intelligence and Data Warehouse applications
Row Store vs Column Store

### Row Store

<table>
<thead>
<tr>
<th>Account Id</th>
<th>First Name</th>
<th>Last Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>Carp</td>
<td>29</td>
<td>8.50</td>
</tr>
<tr>
<td>87911</td>
<td>Craig</td>
<td>Winn</td>
<td>45</td>
<td>25.00</td>
</tr>
</tbody>
</table>

### Column store

<table>
<thead>
<tr>
<th>Key</th>
<th>First Name</th>
<th>Key</th>
<th>Last Name</th>
<th>Key</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>John</td>
<td>12345</td>
<td>Carp</td>
<td>12345</td>
<td>29</td>
<td>8.50</td>
</tr>
<tr>
<td>87911</td>
<td>Craig</td>
<td>87911</td>
<td>Winn</td>
<td>87911</td>
<td>45</td>
<td>25.00</td>
</tr>
</tbody>
</table>
Conclusions
Conclusions

Heaps – Good for low write rate, high read rate
   Catalogs, Wikis, Content management systems

B-Trees – Works for most everything, Good for high concurrency transactional workloads
   Bank accounts, inventory, Payroll processing, Stock trading, etc

Log structured Merge Trees (LSM) – Good for high write rate with low to moderate read rates
   IOT, Time Series, Customer Service logs, etc.

Fractal Trees – Better write latency than traditional B-Trees, worse sequential operations
   Most of the same places you would want to use B-Trees but heavier write work loads

Column stores – Not good for concurrent operations, Store data in columns not rows
   Use for BI and Data Warehouses
Questions?

My Email Address
Schulz@pythian.com