ADMINISTRIVIA

Final Exam: April 27th @ 12:00pm
→ Three short-essay questions.
→ I will provide sample questions this week.

“Final” Presentations: May 6th @ 1:00pm
→ 10 minutes per group
→ Food and prizes for everyone!

Code Reviews: May 8th @ 11:59pm
→ I will announce group assignments and guidelines next class.
Pivotal has submitted QuickStep to be an Apache Project. They are also working on a distributed version of the system.

https://wiki.apache.org/incubator/QuickstepProposal
TODAY’S AGENDA

Background
Implementation Issues
Real-world Examples
Evaluation
DRAM is expensive, son.

It would be nice if our in-memory DBMS could use cheaper storage.
LARGER-THAN-MEMORY DATABASES

Allow an in-memory DBMS to store/access data on disk without bringing back all the slow parts of a disk-oriented DBMS.

Need to be aware of hardware access methods
→ In-memory Storage = Tuple-Oriented
→ Disk Storage = Block-Oriented
OLAP

OLAP queries generally access the entire table. Thus, there isn’t anything about the workload for the DBMS to exploit that a disk-oriented buffer pool can’t handle.
OLAP queries generally access the entire table. Thus, there isn’t anything about the workload for the DBMS to exploit that a disk-oriented buffer pool can’t handle.
OLAP queries generally access the entire table. Thus, there isn’t anything about the workload for the DBMS to exploit that a disk-oriented buffer pool can’t handle.
OLTP

OLTP workloads almost always have **hot** and **cold** portions of the database.
→ We can assume that txns will almost always access hot tuples.

The DBMS needs a mechanism to move cold data out to disk and then retrieve it if it is ever needed again.
LARGER-THAN-MEMORY DATABASES

In-Memory Index

In-Memory Table Heap
- Tuple #00
- Tuple #01
- Tuple #02
- Tuple #03
- Tuple #04

Cold-Data Storage
LARGER-THAN-MEMORY DATABASES

**In-Memory Index**

**In-Memory Table Heap**

- Tuple #00
- Tuple #01
- Tuple #02
- Tuple #03
- Tuple #04

**Cold-Data Storage**
In-Memory Index

In-Memory Table Heap

Cold-Data Storage

header
Tuple #01
Tuple #03
Tuple #04

LARGER THAN MEMORY DATABASES
LARGER-TAN-MEMORY DATABASES

**In-Memory Index**

**In-Memory Table Heap**

**Cold-Data Storage**

- **header**
- Tuple #01
- Tuple #03
- Tuple #04
LARGER-THAN-MEMORY DATABASES

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

Evicted Tuple Block

header
Tuple #01
Tuple #03
Tuple #04
LARGER-TAN-MEMORY DATABASES

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

Evicted Tuple Block
LARGER-TAN-MEMORY DATABASES

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

Evicted Tuple Block

SELECT * FROM table
WHERE id = <Tuple #01>
SELECT * FROM table
WHERE id = <Tuple #01>
LARGER-THAN-MEMORY DATABASES

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

SELECT * FROM table
WHERE id = <Tuple #01>

Evicted Tuple Block
SELECT * FROM table
WHERE id = <Tuple #01>
SELECT * FROM table
WHERE id = <Tuple #01>
OLTP ISSUES

Run-time Operations
→ Cold Tuple Identification

Eviction Policies
→ Timing
→ Evicted Tuple Metadata

Data Retrieval Policies
→ Granularity
→ Retrieval Mechanism
→ Merging back to memory
COLD TUPLE IDENTIFICATION

Choice #1: On-line
→ The DBMS monitors txn access patterns and tracks how often tuples are used.
→ Embed the tracking meta-data directly in tuples.

Choice #2: Off-line
→ Maintain a tuple access log during txn execution.
→ Process in background to compute frequencies.
EVICTION TIMING

Choice #1: Threshold
→ The DBMS monitors memory usage and begins evicting tuples when it reaches a threshold.
→ The DBMS has to manually move data.

Choice #2: OS Virtual Memory
→ The OS decides when it wants to move data out to disk. This is done in the background.
**EVICTED TUPLE METADATA**

**Choice #1: Tombstones**
→ Leave a marker that points to the on-disk tuple.
→ Update indexes to point to the tombstone tuples.

**Choice #2: Bloom Filters**
→ Use approximate data structure for each index.
→ Check both index + filter for each query.

**Choice #3: OS Virtual Memory**
→ The OS tracks what data is on disk. The DBMS does not need to maintain any additional metadata.
EVICTED TUPLE METADATA

In-Memory Index

In-Memory Table Heap

Tuple #00
Tuple #01
Tuple #02
Tuple #03
Tuple #04

Cold-Data Storage
EVICTED TUPLE METADATA

In-Memory Index

Access Frequency

Tuple #00
Tuple #01
Tuple #02
Tuple #03
Tuple #04

In-Memory Table Heap

Tuple #00
Tuple #01
Tuple #02
Tuple #03
Tuple #04

Cold-Data Storage
EVICTED TUPLE METADATA

In-Memory Index

Access Frequency

Tuple #00
Tuple #01
Tuple #02
Tuple #03
Tuple #04

In-Memory Table Heap

Cold-Data Storage
**EVICTED TUPLE METADATA**

**In-Memory Index**

**In-Memory Table Heap**

**Cold-Data Storage**

**Access Frequency**
EVICTED TUPLE METADATA

In-Memory Index

Access Frequency

In-Memory Table Heap

Cold-Data Storage

<table>
<thead>
<tr>
<th>header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple #01</td>
</tr>
<tr>
<td>Tuple #03</td>
</tr>
<tr>
<td>Tuple #04</td>
</tr>
</tbody>
</table>

Tuple #00

Tuple #02

Tuple #00

Tuple #01

Tuple #02

Tuple #03

Tuple #04

Tuple #05
EVICTED TUPLE METADATA

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

header
TUPLE #01
TUPLE #03
TUPLE #04
EVICTED TUPLE METADATA

**In-Memory Index**

**In-Memory Table Heap**

- Tuple #00
- Tuple #02

**Cold-Data Storage**

- header
- Tuple #01
- Tuple #03
- Tuple #04

In-Memory Table Heap:

- Tuple #00
- Tuple #02

Cold-Data Storage:

- header
- Tuple #01
- Tuple #03
- Tuple #04

Diagram showing the relationships between in-memory index, table heap, and cold-data storage.
EVICTED TUPLE METADATA

In-Memory Index

In-Memory Table Heap

Cold-Data Storage

Bloom Filter

Tuple #00

Tuple #02

header

Tuple #01

Tuple #03

Tuple #04

Index
DATA RETRIEVAL GRANULARITY

Choice #1: Only Tuples Needed
→ Only merge the tuples that were accessed by a query back into the in-memory table heap.
→ Requires additional bookkeeping to track holes.

Choice #2: All Tuples in Block
→ Merge all the tuples retrieved from a block regardless of whether they are needed.
→ More CPU overhead to update indexes.
→ Tuples are likely to be evicted again.
RETRIEVAL MECHANISM

Choice #1: Abort-and-Restart
→ Abort the txn that accessed the evicted tuple.
→ Retrieve the data from disk and merge it into memory with a separate background thread.
→ Restart the txn when the data is ready.
→ Cannot guarantee consistency for large queries.

Choice #2: Synchronous Retrieval
→ Stall the txn when it accesses an evicted tuple while the DBMS fetches the data and merges it back into memory.
MERGING THRESHOLD

Choice #1: Always Merge
→ Retrieved tuples are always put into table heap.

Choice #2: Merge Only on Update
→ Retrieved tuples are only merged into table heap if they are used in an UPDATE query.
→ All other tuples are put in a temporary buffer.

Choice #3: Selective Merge
→ Keep track of how often each block is retrieved.
→ If a block’s access frequency is above some threshold, merge it back into the table heap.
REAL-WORLD IMPLEMENTATIONS

H-Store – Anti-Caching
Hekaton – Project Siberia
EPFL’s VoltDB Prototype
Apache Geode – Overflow Tables
MemSQL – Columnar Tables
H-STORE – ANTI-CACHING

On-line Identification
Administrator-defined Threshold
Tombstones
Abort-and-restart Retrieval
Block-level Granularity
Always Merge
HEKATON – PROJECT SIBERIA

Off-line Identification
Administrator-defined Threshold
Bloom Filters
Synchronous Retrieval
Tuple-level Granularity
Always Merge
Off-line Identification

**OS Virtual Memory**

Synchronous Retrieval

Page-level Granularity

Always Merge
EPFL VOLTBDB

In-Memory Table Heap

Tuple #00
Tuple #01
Tuple #02

Cold-Data Storage
In-Memory Table Heap

- Tuple #00
- Tuple #01
- Tuple #02

Cold-Data Storage

EPFL VOLTDDB

Hot Tuples

Cold Tuples
EPFL VOLTDB

In-Memory Table Heap

Cold-Data Storage

Hot Tuples

Cold Tuples

Tuple #00

Tuple #01

Tuple #02
Hot Tuples

In-Memory Table Heap

- Tuple #00
- Tuple #02
- Tuple #01

Cold Tuples

Cold-Data Storage

EPFL VOLTDB
EPFL VOLTDB

In-Memory Table Heap

Hot Tuples
- Tuple #00
- Tuple #02
- Tuple #03

Cold Tuples

Cold-Data Storage
- Tuple #01
EPFL VOLTDB

In-Memory Table Heap

Cold-Data Storage

Hot Tuples

- Tuple #00
- Tuple #03
- Tuple #02

Cold Tuples

- Tuple #01
EPFL VOLTDB

**In-Memory Table Heap**

- Tuple #00
- Tuple #01
- Tuple #02
- Tuple #03

**Cold Tuples**

- Tuple #01
- Tuple #02

**Hot Tuples**

**Cold-Data Storage**
EPFL VOLTDDB

In-Memory Table Heap

- Tuple #00
- Tuple #03
- Tuple #02

Cold-Data Storage

- Tuple #01
- Tuple #01

Hot Tuples

Cold Tuples
APACHE GEODE – OVERFLOW TABLES

On-line Identification
Administrator-defined Threshold
Tombstones (?)
Synchronous Retrieval
Tuple-level Granularity
Merge Only on Update (?)

Source: Apache Geode Documentation
MEMSQL – COLUMNAR TABLES

Administrator manually declares a table as a distinct disk-resident columnar table.
→ Appears as a separate logical table to the application.
→ Uses `mmap` to manage buffer pool.
→ Pre-computed aggregates per block always in memory.

Manual Identification
No Evicted Metadata is needed.
Synchronous Retrieval
Always Merge

Source: MemSQL Documentation
EVALUATION

Compare different design decisions in H-Store with anti-caching.

Storage Devices:
→ Hard-Disk Drive (HDD)
→ Shingled Magnetic Recording Drive (SMR)
→ Solid-State Drive (SSD)
→ 3D XPoint (3DX)
→ Non-volatile Memory (NVRAM)
**MICROBENCHMARK**

10m Tuples – 1KB each
50% Reads / 50% Writes – Synchronization Enabled

Latency (nanosec)

- **1KB Read**
- **1KB Write**
- **64KB Read**
- **64KB Write**

- HDD
- SMR
- SSD
- 3D XPoint
- NVRAM
- DRAM

![Bar chart showing latency for different storage options.](chart.png)
MICROBENCHMARK

10m Tuples - 1KB each
50% Reads / 50% Writes - Synchronization Enabled

- 1KB Read
- 1KB Write
- 64KB Read
- 64KB Write

Latency (nanosec)

HDD
SMR
SSD
3D XPoint
NVRAM
DRAM
MICROBENCHMARK

10m Tuples – 1KB each
50% Reads / 50% Writes – Synchronization Enabled

Latency (nanosec)

1KB Read  1KB Write  64KB Read  64KB Write

Latency (nanosec)

10^0  10^2  10^4  10^6  10^8

HDD  SMR  SSD  3D XPoint  NVRAM  DRAM
MERGING THRESHOLD

YCSB Workload – 90% Reads / 10% Writes
10GB Database using 1.25GB Memory

- Merge (Update-Only)
- Merge (Top-5%)
- Merge (Top-20%)
- Merge (All)

Throughput (txn/sec)

0 50000 100000 150000 200000 250000

HDD (AR)  HDD (SR)  SMR (AR)  SMR (SR)  SSD  3DX  NVMRAM
MERGING THRESHOLD

YCSB Workload – 90% Reads / 10% Writes
10GB Database using 1.25GB Memory

- Merge (Update-Only)
- Merge (Top-5%)
- Merge (Top-20%)
- Merge (All)

Throughput (txn/sec)

Throughput (txn/sec) vs. Storage Technology:
- HDD (AR)
- HDD (SR)
- SMR (AR)
- SMR (SR)
- SSD
- 3DX
- NVMRAM
MERGING THRESHOLD

YCSB Workload – 90% Reads / 10% Writes
10GB Database using 1.25GB Memory

- Merge (Update-Only)
- Merge (Top-5%)
- Merge (Top-20%)
- Merge (All)

Throughput (txn/sec)

HDD (AR)    HDD (SR)    SMR (AR)    SMR (SR)    SSD    3DX    NVMRAM

YCSB Workload – 90% Reads / 10% Writes
10GB Database using 1.25GB Memory
**CONFIGURATION COMPARISON**

**Generic Configuration**
→ Abort-and-Restart Retrieval
→ Merge (All) Threshold
→ 1024 KB Block Size

**Optimized Configuration**
→ Synchronous Retrieval
→ Top-5% Merge Threshold
→ Block Sizes (HDD/SMR - 1024 KB) (SSD/3DX - 16 KB)
TATP BENCHMARK

Optimal Configuration per Storage Device
1.25GB Memory

- HDD
- SMR
- SSD
- 3D XPoint
- NVRAM

Throughput (txn/sec)

<table>
<thead>
<tr>
<th>Storage Device</th>
<th>Generic</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D XPoint</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVRAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DRAM**
VOTER BENCHMARK

Optimal Configuration per Storage Device
1.25GB Memory

Generic  Optimized

Throughput (txn/sec)

DRAM

HDD  SMR  SSD  3DX  NVRAM
PARTING THOUGHTS

Today was about working around the block-oriented access and slowness of secondary storage.

Fast & cheap byte-addressable NVM will make this lecture unnecessary.
Non-Volatile Memory
Project #3 Code Reviews