DATABASE TALKS

Apache Heron / Streamlio
→ Karthik Ramasamy (Founder)
→ Thu Sept 21 @ 12pm (CIC 4th Floor)
→ [More Info]

CockroachDB
→ Ben Darnell (CTO/Founder)
→ Mon Sept 25 @ 4:30pm (GHC 8102)
→ [More Info]
ADMINISTRIVIA

**Homework #2** is due Wednesday September 20\(^{th}\) @ 11:59pm

**Project #1** is due Wednesday October 2\(^{nd}\) @ 11:59pm

**Homework #3** is due Wednesday October 4\(^{th}\) @ 11:59pm
DATABASE STORAGE

Problem #1: How the DBMS represents the database in files on disk.

Problem #2: How the DBMS manages its memory and move data back-and-forth from disk.
DATABASE STORAGE

Spatial Control:
→ Where to write pages on disk.

Temporal Control:
→ When to read pages into memory, and when to write them to disk.
AGAIN, WHY NOT USE THE OS?

Last class I talked about using the OS's memory-mapped files for managing the DBMS memory.

→ The DBMS does not have complete control over what gets evicted.
→ The OS stalls a thread whenever it touches a page not in memory.
AGAIN, WHY NOT USE THE OS?

There are some solutions to this problem:

→ **madvise**: Tell the OS how you expect to read certain pages.
→ **mlock**: Tell the OS that memory ranges cannot be paged out.
→ **msync**: Tell the OS to flush memory ranges out to disk.
AGAIN, WHY NOT USE THE OS?

These APIs are not portable.

Still doesn't stop the OS from blocking our thread.

Tricky to make sure that the OS orders your writes correctly.
BUFFER POOL

The DBMS always knows better, so we will want to manage memory ourselves.

The buffer pool is an in-memory cache of pages read from disk.
LOCKS VS. LATCHES

Locks:
→ Protects the database's logical contents from other transactions.
→ Held for transaction duration.
→ Need to be able to rollback changes.

Latches:
→ Protects the critical sections of the DBMS's internal data structure from other threads.
→ Held for operation duration.
→ Do not need to be able to rollback changes.
TODAY'S AGENDA

Buffer Pool Manager
Replacement Policies
Allocation Policies
Other Memory Pools
BUFFER POOL ORGANIZATION

Memory region organized as an array of fixed-size pages. An array entry is called a frame.

When the DBMS requests a page, an exact copy is placed into one of these frames.
BUFFER POOL ORGANIZATION

Memory region organized as an array of fixed-size pages.
An array entry is called a frame.

When the DBMS requests a page, an exact copy is placed into one of these frames.
BUFFER POOL META-DATA

The page table keeps track of pages that are currently in memory.

Also maintains additional metadata per page:
→ Dirty Flag
→ Pin Counter
BUFFER POOL META-DATA

The page table keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:
→ Dirty Flag
→ Pin Counter
BUFFER POOL META-DATA

The page table keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:
→ Dirty Flag
→ Pin Counter
MULTIPLE BUFFER POOLS

The DBMS doesn't always have a single buffer pool.
→ Multiple buffer pool instances
→ Per-database buffer pool
→ Per-page type buffer pool

Helps reduce latch contention and improve locality.
PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.
→ Sequential Scans
→ Index Scans
The DBMS can also prefetch pages based on a query plan.

→ Sequential Scans
→ Index Scans
PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.
→ Sequential Scans
→ Index Scans
PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.
→ Sequential Scans
→ Index Scans
The DBMS can also prefetch pages based on a query plan.
→ Sequential Scans
→ Index Scans
The DBMS can also prefetch pages based on a query plan.

→ Sequential Scans
→ Index Scans
PRE-FETCHING

Disk Pages

Q1

index-page0

index-page1

index-page2

index-page3

index-page4

index-page5

index-page6

Buffer Pool

index-page0

index-page0
PRE-FETCHING

Disk Pages

index-page0
index-page1
index-page2
index-page3
index-page4
index-page5
index-page6

Buffer Pool

index-page0
index-page1

Q1
PRE-FETCHING

Disk Pages

index-page0
index-page1
index-page2
index-page3
index-page4
index-page5
index-page6

Buffer Pool

index-page0
index-page1

Q1
SCAN SHARING

Queries are able to reuse data retrieved from storage or operator computations. → This is different from result caching.

Allow multiple queries to attach to a single cursor that scans a table. → Queries do not have to be exactly the same. → Can also share intermediate results.
SCAN SHARING

If a query starts a scan and if there one already doing this, then the DBMS will attach to the second query's cursor.

→ The DBMS keeps track of where the second query joined with the first so that it can finish the scan when it reaches the end of the data structure.

Fully supported in IBM DB2 and MSSQL. Oracle only supports cursor sharing for identical queries.
Q1 \textbf{SELECT SUM(val) FROM A}

**SCAN SHARING**

**Buffer Pool**
- page0

**Disk Pages**
- page0
- page1
- page2
- page3
- page4
- page5
SELECT SUM(val) FROM A

Buffer Pool

Disk Pages

page0
page1
page2
page3
page4
page5

SCAN SHARING
SCAN SHARING

Q1: \text{SELECT SUM(val) FROM A}

Buffer Pool
- page3
- page1
- page2

Disk Pages
- page0
- page1
- page2
- page3
- page4
- page5
SCAN SHARING

Q1: SELECT SUM(val) FROM A

Q2: SELECT AVG(val) FROM A

Buffer Pool
- page3
- page1
- page2

Disk Pages
- page0
- page1
- page2
- page3
- page4
- page5
SCAN SHARING

Q1: SELECT SUM(val) FROM A

Q2: SELECT AVG(val) FROM A

Buffer Pool
- page3
- page1
- page2

Disk Pages
- page0
- page1
- page2
- page3
- page4
- page5
**SCAN SHARING**

**Q1**
```
SELECT SUM(val) FROM A
```

**Q2**
```
SELECT AVG(val) FROM A
```

---

**Buffer Pool**
- page3
- page4
- page5

**Disk Pages**
- page0
- page1
- page2
- page3
- page4
- page5

---

**Q2 → Q1**
SCAN SHARING

Q1: SELECT SUM(val) FROM A

Q2: SELECT AVG(val) FROM A

Buffer Pool:
- page3
- page4
- page5

Disk Pages:
- page0
- page1
- page2
- page3
- page4
- page5
SCAN SHARING

Q1: \( \text{SELECT SUM(val) FROM A} \)

Q2: \( \text{SELECT AVG(val) FROM A} \)

Buffer Pool
- page0
- page1
- page2

Disk Pages
- page0
- page1
- page2
- page3
- page4
- page5
BUFFER REPLACEMENT POLICIES

When the DBMS needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool.

Goals:
→ Correctness
→ Accuracy
→ Speed
→ Meta-data overhead
LEAST-RECENTLY USED

Maintain a timestamp of when each page was last accessed.

When the DBMS needs to evict a page, select the one with the oldest timestamp.
→ Keep the pages in sorted order to reduce the search time on eviction.
Approximation of LRU without needing a separate timestamp per page.

→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.

→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.
→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.
→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.

→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.

→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.
→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
CLOCK

Approximation of LRU without needing a separate timestamp per page.

→ Each page has a reference bit.
→ When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":
→ Upon sweeping, check if a page's bit is set to 1.
→ If yes, set to zero. If no, then evict.
PROBLEMS

LRU and CLOCK are susceptible to sequential flooding.

The most recently used page is actually the most unneeded page.
SEQUENTIAL FLOODING

Q1: SELECT * FROM A WHERE id = 1

Buffer Pool

Disk Pages

page0
page1
page2
page3
page4
page5
**SEQUENTIAL FLOODING**

**Q1**

```
SELECT * FROM A WHERE id = 1
```

**Q2**

```
SELECT AVG(val) FROM A
```

---

**Disk Pages**

- page0
- page1
- page2
- page3
- page4
- page5

---

**Buffer Pool**

- page0
**SEQUENTIAL FLOODING**

Q1  SELECT * FROM A WHERE id = 1

Q2  SELECT AVG(val) FROM A

---

**Disk Pages**

- page0
- page1
- page2
- page3
- page4
- page5

**Buffer Pool**

- page0
- page1
- page2
SELECT * FROM A WHERE id = 1

SELECT AVG(val) FROM A

SELECT * FROM A WHERE id = 1
**SEQUENTIAL FLOODING**

Q1: `SELECT * FROM A WHERE id = 1`

Q2: `SELECT AVG(val) FROM A`

Q3: `SELECT * FROM A WHERE id = 1`

---

**Buffer Pool**

- page3
- page1
- page2

**Disk Pages**

- page0
- page1
- page2
- page3
- page4
- page5
BETTER SOLUTIONS

**LRU-K**: Take into account history of the last $K$ references

**Priority Hints**: Allow txns to tell the buffer pool whether a page is important or not.

**Localization**: Choose pages to evict on a per txn/query basis.
Global Policies:
→ Make decisions for all active txns.

Local Policies:
→ Allocate frames to a specific txn without considering the behavior of concurrent txns.
→ Still need to support sharing pages.
OTHER MEMORY POOLS

The DBMS needs memory for things other than just tuples and indexes.

These other memory pools not always backed by disk.
→ Sorting + Join Buffers
→ Query Caches
→ Maintenance Buffers
→ Log Buffers
→ Dictionary Caches
CONCLUSION

The DBMS can manage that sweet, sweet memory better than the OS.
NEXT CLASS

Open Hashing
Extendible Hashing
Linear Hashing
Cuckoo Hashing