Lecture 7: Buffer Management

CREATING THE NEXT®

Administrivia

- To accommodate students who faced challenges with setting up the virtual machine and/or getting familiar with C++, we have extended the deadline for Assignment 1 to Sep 17th @ 11:59pm.
- Enter the cumulative number of slip days used and your team mates at the start of your report.md.

Today's Agenda

Contents

Guidelines

- You can directly run the tests using: ./build/test/external_sort_test
- For debugging, use: gdb ./build/test/external_sort_test
- Team collaboration:
  - Explain your code to your team-mate to see if they know why it doesn't work.
  - Help your team-mate debug if they've run into a wall.

Recap

Data Representation

- `INTEGER/BIGINT/SMALLINT/TINYINT`
  - C/C++ Representation
- `FLOAT/REAL` vs. `NUMERIC/DECIMAL`
  - IEEE-754 Standard / Fixed-point Decimals
- `VARCHAR/VARBINARY/TEXT/BLOB`
  - Header with length, followed by data bytes.
- `TIME/DATE/TIMESTAMP`
  - 32/64-bit integer of (micro)seconds since Unix epoch

Workload Characterization

- On-Line Transaction Processing (OLTP)
  - Fast operations that only read/update a small amount of data each time.
  - OLTP Data Silos
- On-Line Analytical Processing (OLAP)
  - Complex queries that read a lot of data to compute aggregates.
  - OLAP Data Warehouse
- Hybrid Transaction + Analytical Processing
  - OLTP + OLAP together on the same database instance

Database Storage

- Problem 1: How the DBMS represents the database in files on disk.
- Problem 2: How the DBMS manages its memory and moves data back-and-forth from disk.

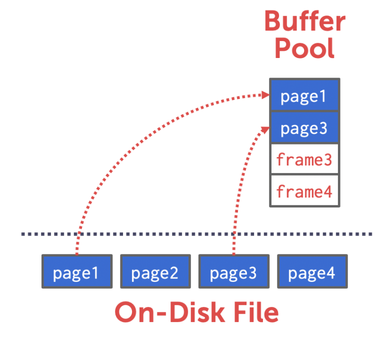
Buffer Pool Manager

Database Storage

- **Spatial** Control:
  - Where to write pages on disk.
  - The goal is to keep pages that are used together often as physically close together as possible on disk.
- **Temporal** Control:
  - When to read pages into memory, and when to write them to disk.
  - The goal is minimize the number of stalls from having to read data from disk.

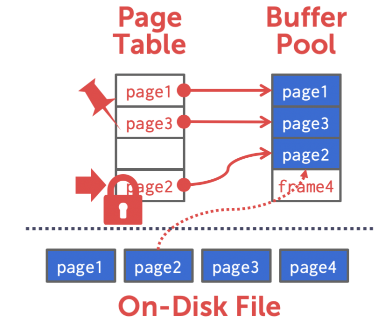
Buffer Pool Organization

- 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 of the data on disk 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 meta-data per page:
  - **Dirty Flag**
  - **Pin/Reference Counter**



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.
  - C++: std::mutex

Page Table vs. Page Directory

- The **page directory** is the mapping from page ids to page locations in the database files.
  - All changes must be recorded on disk to allow the DBMS to find on restart.
- The **page table** is the mapping from page ids to a copy of the page in buffer pool frames.
  - This is an in-memory data structure that does **not** need to be stored on disk.

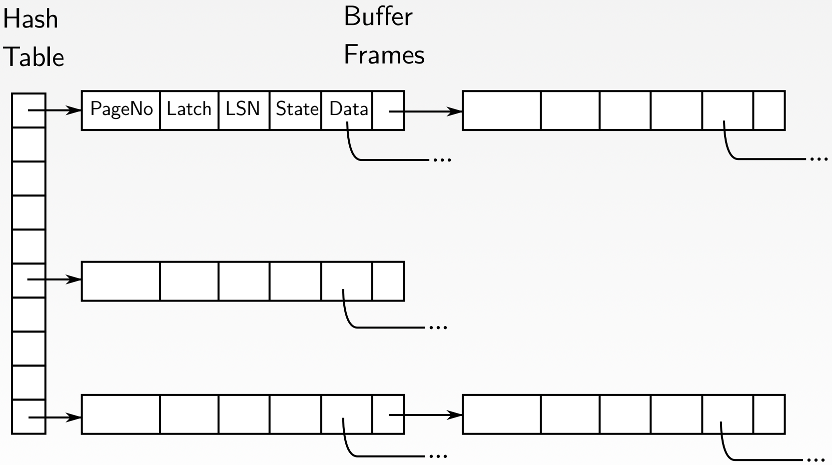
Buffer Manager Interface

Basic interface:

1. FIX (uint64_t page_id, bool is_shared)
2. UNFIX (uint64_t page_id, bool is_dirty)

Pages can only be accessed (or modified) when they are **fixed** in the buffer pool.

Buffer Manager Implementation



The buffer manager itself is protected by one or more **latches**.

Buffer Frame

Maintains the state of a certain page within the buffer pool.

pageNo	the page number
latch	a read/writer latch to protect the page (note: must **not** block access to unrelated pages!)
LSN	LSN of the last change to the page, for recovery (buffer manager must force the log record containing the changes to disk before v
state	clean/dirty/newly created etc.
data	the actual data contained on the page

(will usually contain extra information for buffer replacement)

Usually kept in a hash table.

Buffer Pool Optimizations

Buffer Pool Optimizations

- Multiple Buffer Pools
- Pre-Fetching
- Scan Sharing
- Buffer Pool Bypass
- Background Writing
- Other Pools

Multiple Buffer Pools

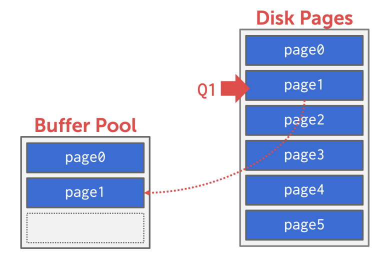
- The DBMS does not always have a single buffer pool for the entire system.
  - Multiple buffer pool instances
  - Per-database buffer pool
  - Per-page type buffer pool
- Helps reduce **latch contention** and improve **locality**. Why?

Multiple Buffer Pools

- Approach 1: Object Id
  - Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.
  - Example: <object_id, page_id, slot_number>
  - ObjectId $\longrightarrow$ Buffer Pool Number
- Approach 2: Hashing
  - Hash the page id to select whichbuffer pool to access.
  - Example: HASH(page_id) % (Number of Buffer Pools)

Pre-Fetching: Sequential Scans

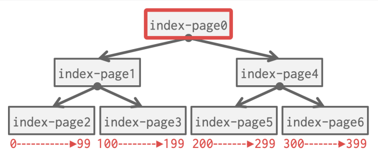
- The DBMS can prefetch pages based on a query plan.
  - Sequential Scans



Pre-Fetching: Index Scans

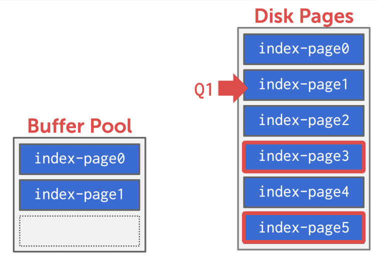
- The DBMS can prefetch pages based on a query plan.
  - Index Scans

```
SELECT *
FROM A
WHERE val BETWEEN 100 AND 250;
```



Pre-Fetching: Index Scans

- The DBMS can prefetch pages based on a query plan.
  - Index Scans




```
SELECT *
FROM A
WHERE val BETWEEN 100 AND 250;
```

Scan Sharing

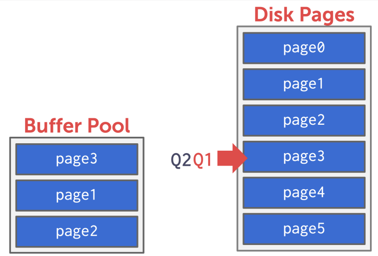
- Queries can **reuse data** retrieved from storage or operator computations.
  - This is different from **result caching**.
- Allow multiple queries to attach to a single cursor that scans a table.
  - Queries do not have to be exactly the same.
  - Can also share intermediate results.

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.

Scan Sharing

```
Q1: SELECT SUM(val) FROM A;
Q2: SELECT AVG(val) FROM A;
Q3: SELECT AVG(val) FROM A LIMIT 100;
```



Buffer Pool Bypass

- The sequential scan operator will not store fetched pages in the buffer pool to avoid overhead.
  - Memory is local to running query.
  - Works well if operator needs to read a large sequence of pages that are contiguous on disk. What is it called?
  - Can also be used for temporary data (sorting, joins).
- Called **light scans** in Informix.

OS Page Cache

- Most disk operations go through the OS API.
- Unless you tell it not to, the OS maintains its own filesystem cache.
- Most DBMSs use direct I/O (`O_DIRECT`) to bypass the OS's cache.
  - Redundant copies of pages.
  - Different eviction policies.

Background Writing

- The DBMS can periodically walk through the page table and write dirty pages to disk.
- When a dirty page is safely written, the DBMS can either evict the page or just unset the dirty flag.
- Need to be careful that we don't write dirty pages before their **log records** have been written to disk.

Other Memory Pools

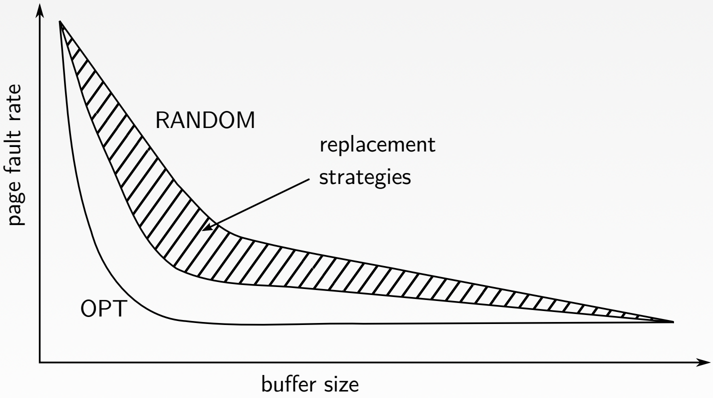
- The DBMS needs memory for things other than just tuples and indexes.
- These other memory pools may not always backed by disk. Depends on implementation.
  - Sorting + Join Buffers
  - Query Caches
  - Maintenance Buffers
  - Log Buffers
  - Dictionary Caches

Buffer Replacement Policies

Buffer Replacement

- 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
- Page State:
  - clean pages can be simply discarded
  - dirty pages have to be written back first

Buffer Replacement Policies



Buffer Replacement Policy - FIFO

First In - First Out (FIFO)

- Simple replacement strategy
- Buffer frames are kept in a linked list (queue)
- Pages inserted at the end, removed from the head
- Keeps the pages that were most recently added to the buffer pool

Does **not** retain frequently-used pages

Buffer Replacement Policy - LFU

Least Frequently Used (LFU)

- Remember the number of accesses per page
- Infrequently used pages are removed first
- Maintain a priority queue of pages

Sounds plausible, but too expensive in practice.

Buffer Replacement Policy - LRU

Least-Recently Used (LRU)

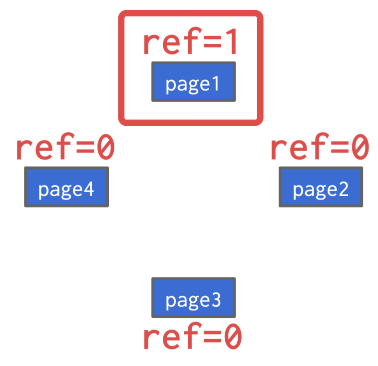
- Maintain a timestamp of when each page was last accessed.
- When the DBMS needs to evict a page, select the one with the **oldest access timestamp**.
  - Keep the pages in sorted order to reduce the search time on eviction.
  - Buffer frames are kept in a double-linked list
  - Remove from the head
  - When a frame is unfixed, move it to the end of the list
  - "Hot" pages are retained in the buffer

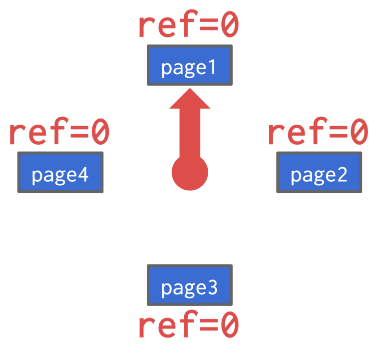
A very popular policy.

Buffer Replacement Policy - CLOCK

- LRU works well, but the LRU list is a hot spot and need meta-data.
- 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.

Buffer Replacement Policy - CLOCK



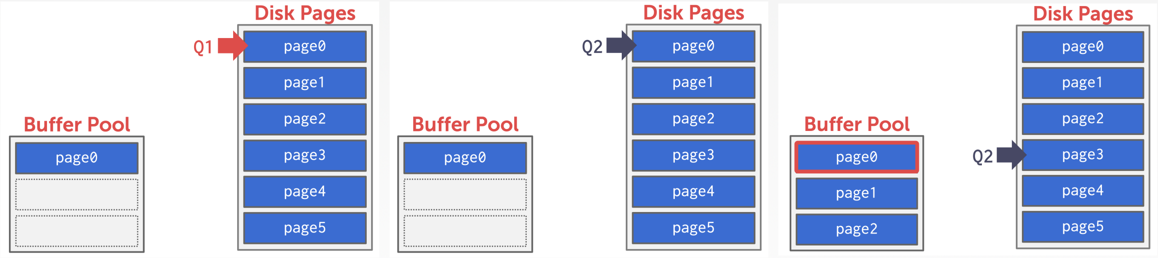


Problems

- LRU and CLOCK replacement policies are susceptible to **sequential flooding**.
  - A query performs a sequential scan that reads every page.
  - This **pollutes** the buffer pool with pages that are read once and then never again.
- The most recently used page is actually the most unneeded page.

```
Q1: SELECT * FROM A WHERE id = 1;
Q2: SELECT AVG(val) FROM A; -- Sequential Scan
```

Sequential Flooding



Better Policies - LRU-K

- Track the history of last K references to each page as timestamps and compute the interval between subsequent accesses.
- The DBMS then uses this history to estimate the next time that page is going to be accessed.
- Degenerates to classic LRU when K = 1
- **Scan resistant** policy

Better Policies - 2Q

Maintain two queues (FIFO and LRU)

- Some pages are accessed only once (*e.g.*, sequential scan)
- Some pages are hot and accessed frequently
- Maintain separate lists for those pages
- **Scan resistant** policy

1. Maintain all pages in FIFO queue
2. When a page that is currently in FIFO is referenced again, upgrade it to the LRU queue
3. Prefer evicting pages from FIFO queue

Hot pages are in LRU, read-once pages in FIFO.

Better Policies - Priority Hints

- The DBMS knows what the context of each page during query execution.
- It can provide hints to the buffer pool on whether a page is important or not.
- 2Q tries to recognize read-once pages
- But the DBMS knows this already!
- It could therefore give **hints** when unfixing
- Example: **will-need** or **will-not-need** hint will determine which queue the page is added to

Georgia Tech

Conclusion

- The DBMS can manage that sweet, sweet memory better than the OS.
- Leverage the semantics about the query plan to make better decisions:
  - Evictions
  - Allocations
  - Pre-fetching

Next Class

- Buffer Management Implementation