CS525: Advanced Database Organization

Notes 3: Database Storage
Part I

Gerald Balekaki

Department of Computer Science

Illinois Institute of Technology

gbalekaki@iit.edu

June 28th 2023

Slides: adapted from a course taught by Andy Pavlo, Carnegie Mellon University

Reading

- `Database Systems: The Complete Book, 2nd Edition,`
  - *Chapter 2: Data Storage*
- `Database System Concepts (6th/7th Edition)`
  - *Chapter 10 (6th)/ Chapter 13 (7th)*

System Design Goals

- We start learning how to build software that manages a database.
- Allow the DBMS to manage databases that exceed the amount of memory available.
- Since reading/writing to disk is expensive, so it must be managed carefully to avoid large stalls and performance degradation.
    - We do not want large stalls from fetching something from disk to slow down everything else.
    - We want the DBMS to be able to process other queries while it is waiting to get the data from disk.
- Random access on disk is usually much slower than sequential access, so the DBMS will want to maximize sequential access.

Disk-Oriented DBMS Overview

- The database is all on disk, and the data in database files is organized into pages, with the first page being the directory page.
- In order to operate on the data, the DBMS needs to bring the data into memory.
- It does this by having a buffer pool that manages the movement of data back and forth between disk and memory.
- The DBMS also has an execution engine that will execute queries.
- The execution engine will ask the buffer pool for a specific page, and the buffer pool will take care of bringing that page into memory and giving the execution engine a pointer to that page in memory.
- The buffer pool manager will ensure that the page is there while the execution engine is operating on that part of memory

Why Not Use the OS?

- DBMS (almost) always wants to control things itself and can do a better job at it since it knows more about the data being accessed and the queries being processed.
  - Flushing dirty pages to disk in the correct order.
  - Specialized prefetching.
  - Buffer replacement policy.
  - Thread/process scheduling.
- The OS is not your friend.
- *Even though the system will have functionalities that seem like something the OS can provide, having the DBMS implement these procedures itself gives it better control and performance.*

Database Storage

- `Problem#1`: How the DBMS represents the database in files on disk
  - i.e., how to lay out data on disk.
- `Problem#2`: How the DBMS manages its memory and move data back-and-forth from disk.

Today's Agenda

- File Storage
- Page Layout
- Tuple Layout

File Storage

- In its most basic form, a DBMS stores a database as files on disk.
- Some may use a file hierarchy, others may use a single file (e.g., SQLite).
  - The OS doesn't know anything about these files.
  - Only the DBMS knows how to decipher their content since it is encoded in a way specific to the DBMS.
  - These files will be stored on the top of file system supported by OS.
- The DBMS will rely on the file system to provide with basic read/write operations.

- Early systems in the 1980s used custom filesystems on raw storage
  - Some enterprise DBMSs still support this.
  - Most newer DBMSs do not do this.

Storage manager

- The DBMS's storage manager is responsible for maintaining a database's files on disk.
    - Some do their own scheduling for reads and writes.
- It organizes the files as a collection of pages.
    - Tracks data read/written to pages.
    - Tracks the available space.

Database Pages

- The DBMS organizes the database across one or more files in fixed-size blocks of data called pages
- A page is a fixed-size block of data.
  - It can contain different kinds of data
    - tuples, meta-data, indexes, log records, . . .
  - Most systems do not mix page types within pages.
  - Some systems require a page to be self-contained.
    - meaning that all the information needed to read each page is on the page itself.
- Each page is given a unique internal identifier called page_id generated by database system.
  - The DBMS uses an indirection layer to map page ids to physical locations (a file path and offset)
    - The upper levels of the system will ask for a specific page number.
    - Then, the storage manager will have to turn that page number into a file and an offset to find the page.

Database Pages

- Most DBMSs uses fixed-size pages to avoid the engineering overhead needed to support variable-sized pages.
    - For example, with variable-size pages, deleting a page could create a hole in files that the DBMS cannot easily fill with new pages.

Database Pages

- There are three different notions of “pages” in a DBMS:
    - Hardware Page (usually 4KB)
        - Different storage devices have different page sizes
    - OS Page (usually 4KB)
    - Database Page (512B-16KB)
        - `SQLite`, `DB2`, `Oracle`: 4KB
        - `SQL Server`, `PostgreSQL`: 8KB
        - `MySQL`: 16KB

Database Pages: Hardware Page: Failsafe Write

- A hardware page is the largest block of data that the storage device can guarantee failsafe writes.
- The storage device guarantees an atomic write of the size of the hardware page.
- If the hardware page is 4 KB and the system tries to write 4 KB to the disk, either all 4 KB will be written, or none of it will.
- This means that if our database page is larger than our hardware page, the DBMS will have to take extra measures to ensure that the data gets written out safely since the program can get partway through writing a database page to disk when the system crashes.

Page Storage Architecture

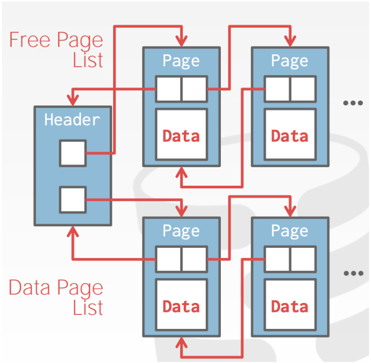
- The DBMS needs a way to find a page on disk given a page id
- Different DBMSs manage pages in files on disk in different ways.
  - Heap File Organization
  - Sequential / Sorted File Organization
  - Hashing File Organization
  - Tree File Organization
- At this point in the hierarchy, we don't need to know anything about what is inside of the pages.

Database Heap

- A heap file is an unordered collection of pages where tuples that are stored in random order.
  - Create / Get / Write / Delete Page
  - Must also support iterating over all pages.
- It is easy to find pages if there is only a single file.
- Need meta-data to keep track of what pages exist in multiple files and which ones have free space.
- Two ways to represent a heap file:
  - Linked List
    - Header page holds pointers to a list of free pages and a list of data pages.
  - Page Directory
    - DBMS maintains special pages that track locations of data pages along with the amount of free space on each page.
- The DBMS can locate a page on disk given a page id by using a linked list of pages or a page directory.

Heap File: Linked List

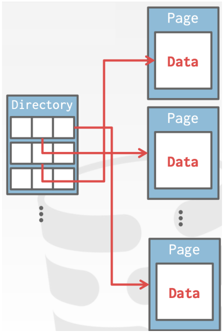
- Maintain a header page at the beginning of the file that stores two pointers:
  - HEAD of the free page list.
  - HEAD of the data page list.
- Each page keeps track of how many free slots they currently have.



- However, if the DBMS is looking for a specific page, it has to do a sequential scan on the data page list until it finds the page it is looking for.

Heap File: Page Directory

- The DBMS maintains special pages that tracks the location of data pages in the database files.
- The directory also records the number of free slots per page.
- The DBMS has to make sure that the directory pages are in sync with the data pages.

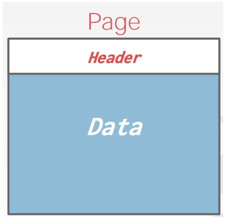


Today's Agenda

- File Storage
- Page Layout
- Tuple Layout

Page Header

- Every page includes a header that records meta-data about the page's contents:
  - Page Size
  - Checksum
  - DBMS Version
  - Transaction Visibility
  - Compression Information
- Some systems require pages to be self-contained (e.g., Oracle).

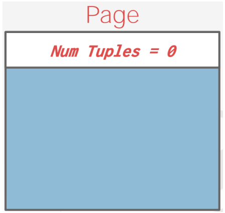


Page Layout

- For any page storage architecture, we now need to understand how to organize the data stored inside of the page.
  - We are still assuming that we are only storing tuples.
- There are two approaches to laying out data in pages:
  - Tuple-oriented
  - Log-structured

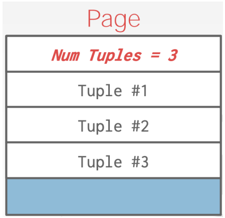
Tuple Storage

- How to store tuples in a page?
- **Strawman Idea**: Keep track of the number of tuples in a page and then just append a new tuple to the end.



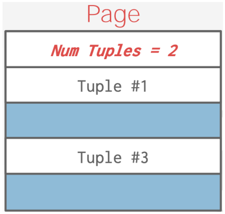
Tuple Storage

- How to store tuples in a page?
- **Strawman Idea**: Keep track of the number of tuples in a page and then just append a new tuple to the end.



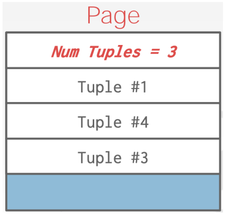
Tuple Storage

- How to store tuples in a page?
- **Strawman Idea**: Keep track of the number of tuples in a page and then just append a new tuple to the end.
  - What happens if we delete a tuple?



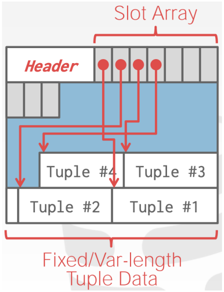
Tuple Storage

- How to store tuples in a page?
- **Strawman Idea**: Keep track of the number of tuples in a page and then just append a new tuple to the end.
    - What happens if we delete a tuple?
    - What happens if we have a variable-length attribute? Is this work?



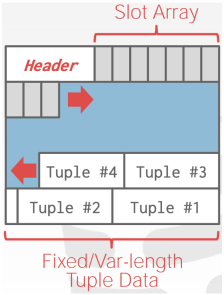
Slotted Pages

- The most common layout scheme is called slotted pages.
- Slot array, which keeps track of the location of the start of each tuple.
  - The slot array maps "slots" to the tuples' starting position offsets.
    - Page maps slots to offsets
- The header keeps track of:
  - The # of used slots,
  - The offset of the starting location of the last slot used, and
  - a slot array, which track of the location of the start of each tuple



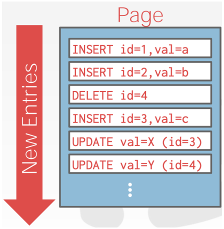
Slotted Pages

- To add a tuple, the slot array will grow from the beginning to the end, and the data of the tuples will grow from end to the beginning.
- The page is considered full when the slot array and the tuple data meet.



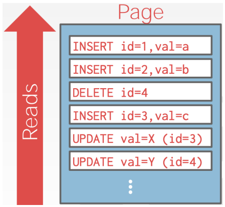
Log-Structured File Organization

- Instead of storing tuples in pages, the DBMS only stores log records (store info how that tuple was created or modified)
- The system appends log records to the file of how the database was modified (insert, update, delete)
  - Inserts store the entire tuple.
  - Deletes mark the tuple as deleted.
  - Updates contain the delta of just the attributes that were modified.
- i.e., stores records to file of how the database was modified (insert, update, deletes).



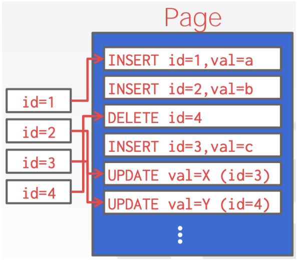
Log-Structured File Organization

- To read a record, the DBMS scans the log backwards and "recreates" the tuple to find what it needs.
- Fast writes, potentially slow reads.
- Works well on append-only storage because the DBMS can not go back and update the data.



Log-Structured File Organization

- To avoid long reads, the DBMS can have indexes to allow it to jump to specific locations in the log



Log-Structured File Organization

- Periodically compact the log.
  - if it had a tuple and then made an update to it, it could compact it down to just inserting the updated tuple
  - The issue with compaction is the DBMS ends up with write amplification (it re-writes the same data over and over again).
- `Casandra, HBase, LevelDB`

Today's Agenda

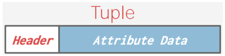
- File Storage
- Page Layout
- Tuple Layout

Tuple Layout

- A tuple is essentially a sequence of bytes.
- It’s the job of the DBMS to interpret those bytes into attribute types and values.

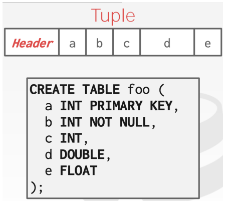
Tuple Layout

- Tuple Header: Contains meta-data about the tuple.
- Each tuple is prefixed with a header that contains meta-data about it.
  - Visibility info (concurrency control)
    - i.e., information about which transaction created/modified that tuple
  - Bit Map for NULL values.
    - indicates which attributes of a tuple has a Null value
- Note that the DBMS does not need to store meta-data about the schema of the database here.



Tuple Data

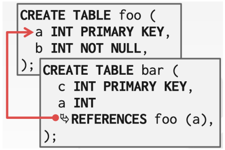
- Tuple Data: Actual data for attributes.
- Attributes are typically stored in the order that you specify them when you create the table.
- This is done for software engineering reasons.
- Most DBMSs do not allow a tuple to exceed the size of a page.



```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
  c INT,
  d DOUBLE,
  e FLOAT
);
```

Denormalized Tuple Data

- If two tables are related, the DBMS can “pre-join” them, so the tables end up on the same page
- Can physically denormalize (e.g., “pre-join”) related tuples and store them together in the same page

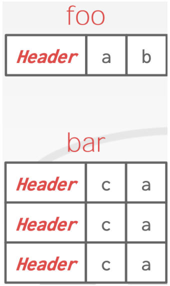


```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
```

```
CREATE TABLE bar (
  c INT PRIMARY KEY,
  a INT
    REFERENCES foo (a),
);
```

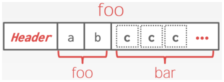
Denormalized Tuple Data

- Can physically denormalize (e.g., “pre-join”) related tuples and store them together in the same page



Denormalized Tuple Data

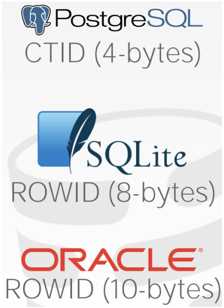
- Potentially reduces the amount of I/O for common workload patterns.
  - DBMS only has to load in one page rather than two separate pages,
- Can make updates more expensive.
  - DBMS needs more space for each tuple



- Not a new idea.
  - IBM System R did this in the 1970s.
  - Several NoSQL DBMSs do this without calling it physical denormalization.

Record ID

- The DBMS needs a way to keep track of individual tuples.
- Each tuple is assigned a unique record identifier.
    - Most common: page_ID + offset/slot, where offset gives location of tuple within page
    - Can also contain file location info.
        - typically, page_ID = FileID + Offset, where Offset gives location of block within file
- An application **cannot** rely on these ids to mean anything.



Conclusion

- Database is organized in pages.
- Different ways to track pages.
- Different ways to store pages.
- Different ways to store tuples.

Next

- Value Representation
- Storage Models