CS525: Advanced Database Organization

Notes 3: Database Storage
Part I

Yousef M. Elmehdwi

Department of Computer Science

Illinois Institute of Technology

yelmehdwi@iit.edu

August 30th 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.
- To work this data, the DBMS needs to bring the data into memory.
- This is achieved through the utilization of a buffer pool, which manages the bi-directional transfer of data between the disk and memory.
- The execution engine executes queries.
- The execution engine requests a particular page from the buffer pool, which then handles the task of fetching the page into memory. Subsequently, the buffer pool provides the execution engine with a memory-based pointer to that requested page.
  execution engine a pointer to that page in memory.
- The buffer pool manager keeps these pages in memory while the execution engine operates on them, ensuring efficient data access and manipulation.

Why Not Use the OS?

- DBMS (almost) tends to prefer managing tasks independently due to its deeper understanding of accessed data and executed queries.
- This approach often yields superior results as compared to relying solely on the operating system.
    - 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

How do we organize the database over pages?

- 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 storage managers even handle their own scheduling for read and write operations.
- It organizes the files as a collection of pages.
  - Tracks data read/written to pages.
  - Tracks the available space.

- *The manager structures files as a set/sequence of pages and monitors data read and written to these pages. Furthermore, it keeps track of the available space within the database.*

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.
    - To map these page IDs to their physical locations, which include file paths and offsets, the DBMS employs an indirection layer.
        - 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/concepts of “pages” in a DBMS:
  - Hardware Page (usually 4KB)
    - This is a physical unit of storage
    - Different storage devices have varying page sizes to manage data efficiently
  - OS Page (usually 4KB)
    - This is a unit that the operating system uses for various management and memory allocation purposes.
  - Database Page (512B-16KB)
    - This is the unit of data storage within the DBMS itself.
    - Database pages can vary in size, typically ranging from 512 bytes (B) to 16KB.
    - Different DBMSs may adopt different sizes for their database pages:
      - `SQLite`, `DB2`, `Oracle`: 4KB
      - `SQL Server`, `PostgreSQL`: 8KB
      - `MySQL`: 16KB

Database Pages

- *These distinctions are important because they influence how data is managed and stored within the DBMS, how it interacts with the operating system and hardware, and how data is retrieved and manipulated during various database operations. The choice of page size can impact factors such as storage efficiency, memory usage, and performance of the DBMS*

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.
- For example, if the hardware page size is 4 KB and the system attempts to write 4 KB of data to the disk, it's ensured that either the entire 4 KB will be successfully written or none of it will be.
- When a database page size exceeds the hardware page size, additional precautions/measures are necessary.
  - This is because a situation can arise where the system crashes midway through writing a database page to disk.
  - In this scenario, the Database Management System (DBMS) needs to take extra steps to ensure data consistency and reliability during write operations.

Page Storage Architecture

- Within a DBMS, the challenge lies in locating a specific page on the disk based on its page id.
    - *The DBMS needs a way to find a page on disk given a page id*
- Various DBMSs adopt distinct strategies for managing pages within disk files.

Page Storage Architecture

- Different DBMSs manage pages in files on disk in different ways.
- Some common methods include:
    - Heap File Organization
        - This involves storing data pages in an unordered manner.
        - It's a straightforward approach where data is appended as it arrives, without any specific ordering.
    - Sequential / Sorted File Organization
        - Data pages are arranged in a sequential or ordered manner based on certain criteria, such as a primary key.
        - This can speed up certain types of queries and enable range-based access.
    - Hashing File Organization
        - Data pages are distributed across the disk using a hash function.
        - This aims to evenly distribute data and optimize retrieval through calculated hash values.
    - Tree File Organization
        - Data pages are structured in tree-like structures (such as B-trees or B+ trees) to facilitate efficient search, insertion, and deletion operations.
- At this point in the hierarchy, we don't need to know anything about what is inside of the pages.

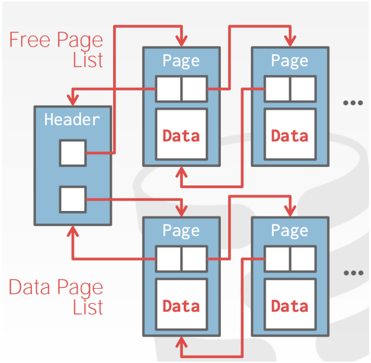
Database Heap

- 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 ensuring comprehensive access to stored data.
- It is easy to find pages if there is only a single file.
- However, when dealing with multiple files, metadata becomes essential to track existing pages and available space.
- Heap files can be represented in two primary ways:
  - Linked List
    - Header page holds pointers to a list of free pages and a list of data pages.
  - Page Directory
    - DBMS maintains dedicated/special pages to track data page locations and available space.
- The DBMS can locate a page on disk given a page id either through linked lists 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.

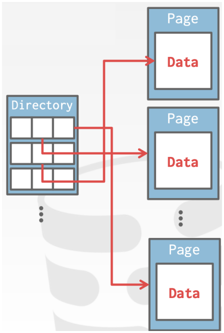
- Each page keeps track of how many free slots they currently have.



- However, when the DBMS needs to locate a specific page, it must conduct a sequential scan across the data page list until the desired page is found.

Heap File: Page Directory

- The DBMS maintains special pages that tracks the location of data pages within the database files.
- The directory also keeps a record of the available free slots on each page.
- The DBMS has to make sure that the directory pages are in sync with the data pages.
    - This guarantees that the information in the page directory accurately reflects the state of the data pages, maintaining data consistency and reliability.



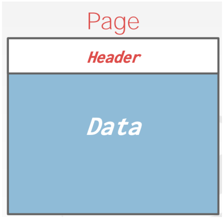
Today's Agenda

- File Storage
- Page Layout
    - *What does look like inside the page?*
- Tuple Layout

Page Header

- Every page includes a header section that records meta-data about the page's contents:
  - Page Size
  - Checksum for data integrity
  - DBMS Version
  - Transaction Visibility details
  - Compression-related Information

- Some systems require pages to be self-contained (e.g., Oracle).
  - *all the required information to understand and interpret the page's contents is present within the page itself.*
  - *ensures data consistency and accessibility, especially in scenarios where pages need to be recovered after system failures.*

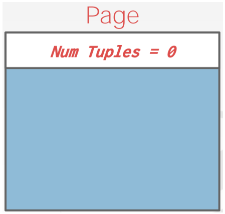


Page Layout

- 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

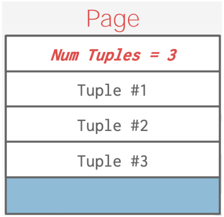
Tuple Storage

- 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.



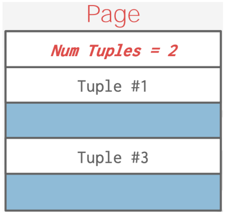
Tuple Storage

- 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.
  - Page header or metadata section stores tuple count on the page.
  - New tuples are added by placing them at the end of the page after existing tuples.
  - This process continues as long as there's sufficient available space.
- *Works great for fixed length tuples.*
- *However, problems arise when tuples are deleted when tuples have variable-length attributes*



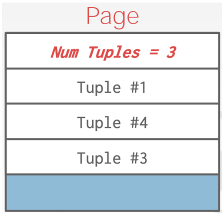
Tuple Storage

- 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?



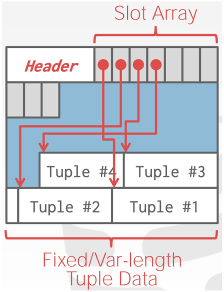
Tuple Storage

- 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?



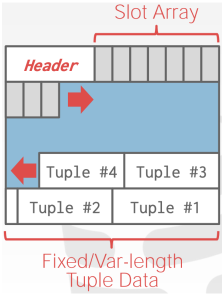
Slotted Pages

- The most common layout scheme is called slotted pages.
- Slot array, which keeps track of the location of the start of each tuple.
- A slot array is employed to maintain the starting position of each tuple on the page.
- Each entry in the slot array corresponds to a slot, mapping to the offset where a tuple begins.
    - *The slot array essentially maps slots to the offsets where tuples start within the page. This enables efficient access to specific tuples.*
- 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

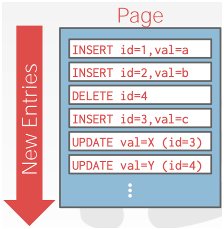
- To add a tuple, the slot array will grow from the beginning to the end, and the data of the tuples will grow from end to the beginning.
- The page is considered full when the slot array and the tuple data meet.



Log-Structured File Organization

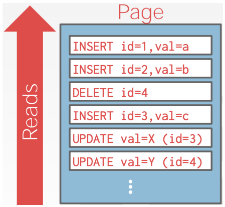
- Instead of storing tuples in pages, the DBMS only stores log records (store info how that tuple was created or modified)
- The system appends log records to the file of how the database was modified (insert, update, delete)
  - Inserts store the entire tuple.
  - Deletes mark the tuple as deleted.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

- This log-structured approach provides a history of all modifications made to the database.



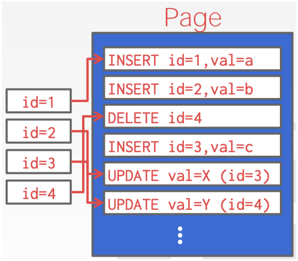
Log-Structured File Organization

- 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.
  - *Since the approach doesn't support direct updates of existing data (it only appends new log records),*



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

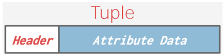
- File Storage
- Page Layout
- Tuple Layout

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

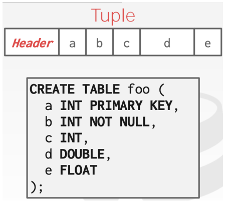
- Tuple Header: Contains meta-data about the tuple.
- Each tuple is prefixed with a header that contains meta-data about it.
  - Visibility info (concurrency control)
    - i.e., information about which transaction created/modified that tuple
  - Bit Map for NULL values.
    - indicates which attributes within the tuple hold Null values.
    - offers a quick reference to the presence of Nulls in the tuple.



- Note that the DBMS does not need to store meta-data about the schema of the database here.

Tuple Data

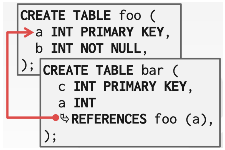
- Tuple Data: Actual data for attributes.
- Attributes are typically stored in the order that you specify them when you create the table.
    - *this consistent order simplifies data retrieval and ensures consistency across tuples.*
- This is done for software engineering reasons.
    - *it simplifies data management, query processing, and maintenance of the data structure.*
- Most DBMSs enforce a limit on the size of a tuple, ensuring it doesn't exceed the size of a page.
    - *this constraint ensures that data can be efficiently managed within the storage system*



```
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, meaning they have a common attribute that links them, 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

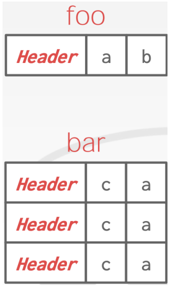


```
CREATE TABLE foo (
  a INT PRIMARY KEY,
  b INT NOT NULL,
);
```

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

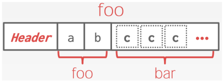
Denormalized Tuple Data

- Can physically denormalize (e.g., “pre-join”) related tuples and store them together in the same page
- *Instead of storing related data in separate pages, the DBMS can decide to physically organize and store related tuples from both tables together on 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.

  - *The terminology might differ across database systems, but the fundamental concept of optimizing data organization to improve query efficiency remains consistent.*

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.

PostgreSQL
CTID (4-bytes)

SQLite
ROWID (8-bytes)

ORACLE
ROWID (10-bytes)

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