Discussion 6

Storage + NoSQL + Project 3 Intro

EECS 484

Logistics

- **Project 2** due **Oct 22nd** at 11:45 PM ET
- **Project 3** released, due **Nov 1st** at 11:45 PM ET

Storage

Storage

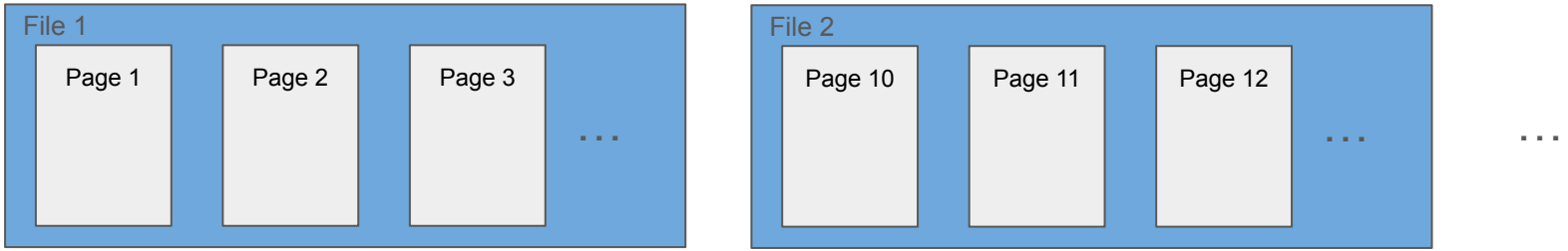
- Volatile Storage
  - Temporary memory that requires power to retain data
  - Byte-addressable and allows random access
    - allowing data to be accessed directly from any location
  - Faster, smaller, and more expensive
  - DRAM, CPU Caches, and CPU Registers
- Non-Volatile Storage
  - Permanent memory that retains data even when power is off
  - Block-addressable and allows sequential access
    - Random access on non-volatile storage is usually much slower than sequential access
  - Slower, larger, and cheaper
  - SSD, HDD, and Network Storage

Storage

- Why is this important?
  - Different types of storage devices have different properties, speeds, and sizes
  - We organize data and design database systems around these properties
- Disk-based Architecture
  - The DBMS assumes a database is primarily stored as one or more files on disk (non-volatile)
    - Retains data even when power is off
    - Larger and cheaper compares to memory
  - The DBMS's components manage the movement of data between non-volatile and volatile storage

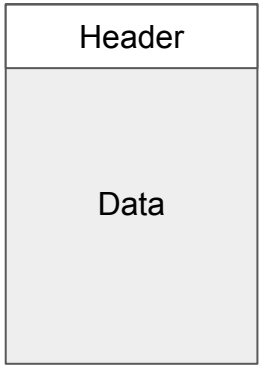
Files and Database Pages

- The DBMS stores a database as one or more files on disk
- Storage manager organizes the files as a collection of pages
  - Track read/write operations for the pages
  - Track available space
- A page is a fixed-size block of data
  - Some contain tuples, some contain metadata, logs, indexes…
  - Each page given unique ID that allows the storage manager to find it



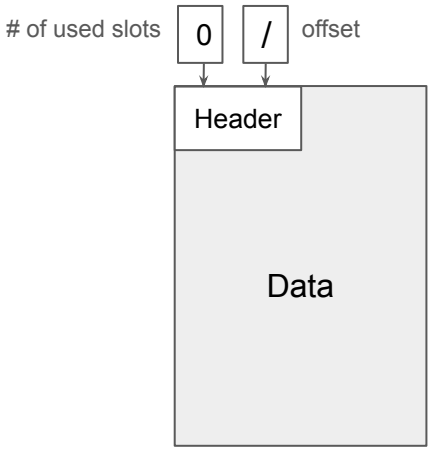
Page Layout

- Page Header
  - Each page contains a header that stores metadata about the page's content
    - Page size, DBMS version, transaction visibility, etc.
  - Some systems require pages to be self-contained
    - Each page contains all the information necessary to interpret and access the data within the page
    - Help the DBMS achieve its goal of safely and correctly storing data



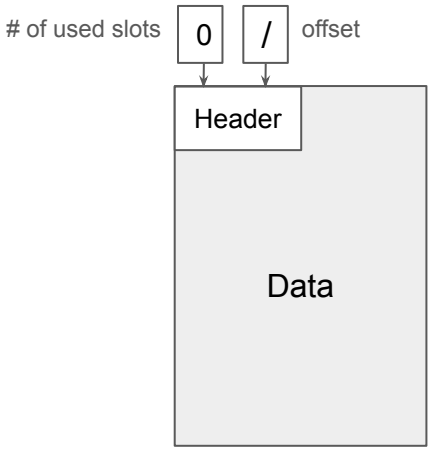
Page Layout

- Slotted pages
  - Data (tuples) are stored in a page using slotted pages
  - The most common layout scheme in DBMS
  - The header of the page keeps track of:
    - The number of used slots
    - The offset of the starting location of the last slot used



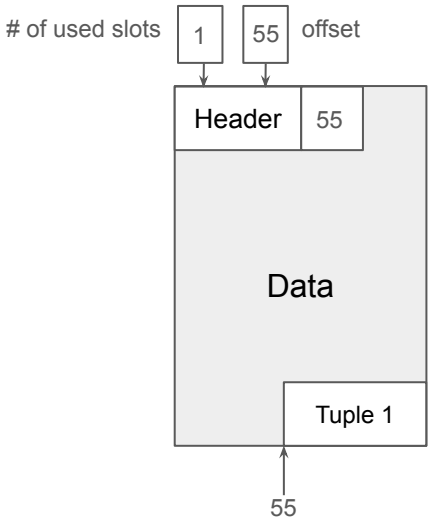
Page Layout

- Slotted pages
  - Data (tuples) are stored in a page using slotted pages
  - The most common layout scheme in DBMS
  - The header of the page keeps track of:
    - The number of used slots
    - The offset of the starting location of the last slot used
  - Slot array
    - maps slots to the tuples' starting position offsets



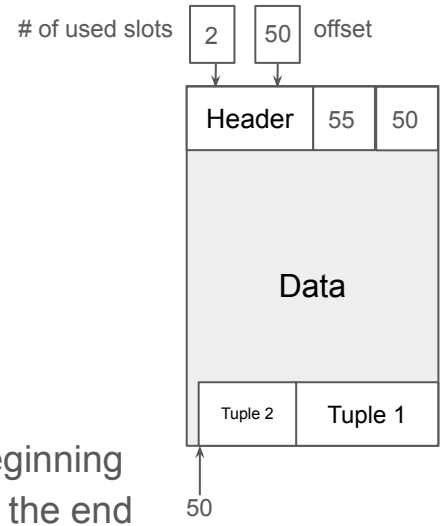
Page Layout

- Slotted pages
  - Data (tuples) are stored in a page using slotted pages
  - The most common layout scheme in DBMS
  - The header of the page keeps track of:
    - The number of used slots
    - The offset of the starting location of the last slot used
  - Slot array
    - maps slots to the tuples' starting position offsets



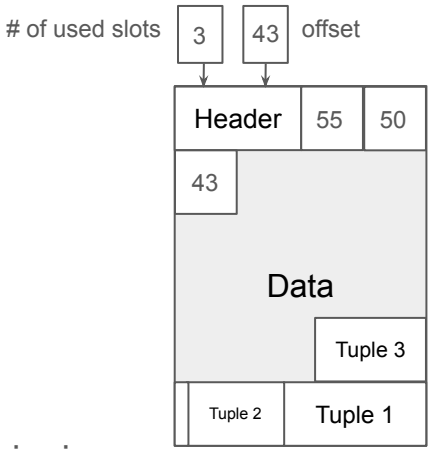
Page Layout

- Slotted pages
  - Data (tuples) are stored in a page using slotted pages
  - The most common layout scheme in DBMS
  - The header of the page keeps track of:
    - The number of used slots
    - The offset of the starting location of the last slot used
  - Slot array
    - maps slots to the tuples' starting position offsets
  - Tuples are allocated from the end of the page toward the beginning
  - The slot array is built from the beginning of the page toward the end



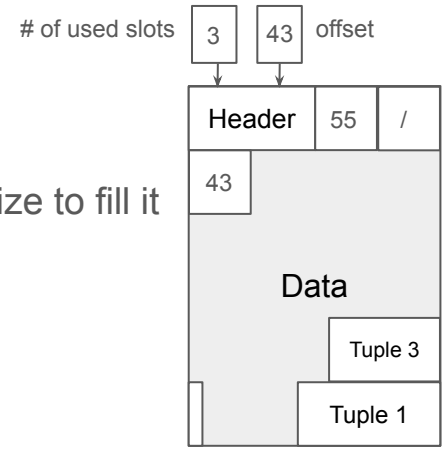
Page Layout

- Slotted pages
  - Data (tuples) are stored in a page using slotted pages
  - The most common layout scheme in DBMS
  - The header of the page keeps track of:
    - The number of used slots
    - The offset of the starting location of the last slot used
  - Slot array
    - maps slots to the tuples' starting position offsets
  - Tuples are allocated from the end of the page toward the beginning
  - The slot array is built from the beginning of the page toward the end
  - The page is considered full when the tuples and the slot array meet in the middle
    - At this point, the system will move to the next available page



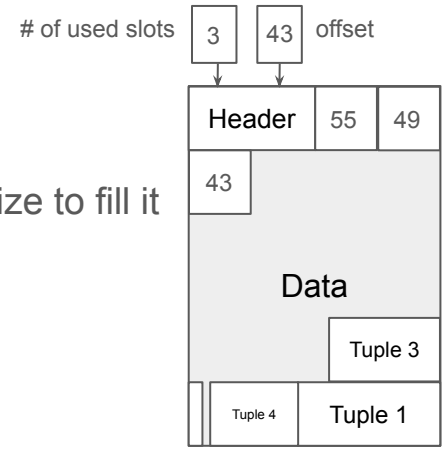
Page Layout

- Slotted pages
  - When a tuple is deleted:
    - The space can be left as it is
      - Waiting for a new tuple of the same or smaller size to fill it



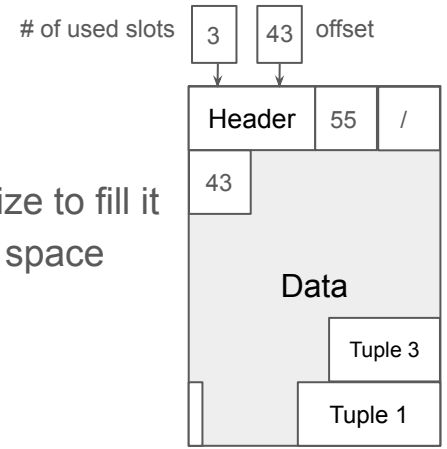
Page Layout

- Slotted pages
  - When a tuple is deleted:
    - The space can be left as it is
      - Waiting for a new tuple of the same or smaller size to fill it



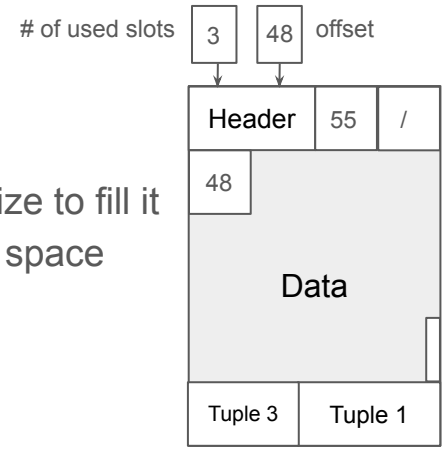
Page Layout

- Slotted pages
  - When a tuple is deleted:
    - The space can be left as it is
      - Waiting for a new tuple of the same or smaller size to fill it
    - or the existing tuples can be rearranged to fill the free space



Page Layout

- Slotted pages
  - When a tuple is deleted:
    - The space can be left as it is
      - Waiting for a new tuple of the same or smaller size to fill it
    - or the existing tuples can be rearranged to fill the free space



Tuples

- The DBMS needs a way to keep track of individual tuples
- Each tuple is assigned a unique record id
  - Most commonly: page_id + offset/slot
  - Can also contain file location information

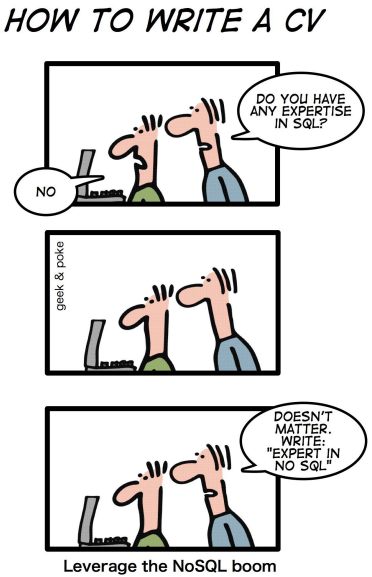
Storage Overview

- The DBMS assumes a database is primarily stored as one or more files on disk
  - Disk: Non-volatile, data is safer there
- Storage manager organizes the files as a collection of pages
- A page is a fixed-size block of data
  - Some contain tuples, some contain metadata, logs, indexes…
  - Each page given unique ID that allows the storage manager to find it
- (Next week) To speed up accessing (read/write) data from pages, we use data structures(hash table/tree)
  - Helps speed up searches instead of scanning all a database's pages for a certain value/tuple

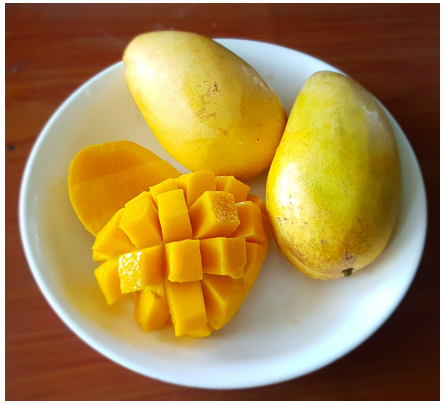
MongoDB and NoSQL

NoSQL

- Not Only SQL!
  - Non relational databases
  - Use very different data structures compared to traditional relational databases
- Reading: https://www.mongodb.com/nosql-explained
- Different data models used by different distributions
- Data in a common set doesn't need to adhere to a schema



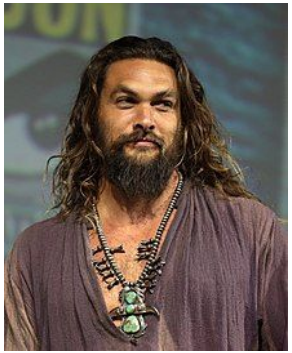
MongoDB



- Instead of rows we have documents
  - JSON objects in JavaScript syntax, consisting of field:value pairs
    - I.E “Age”: “7”
  - In addition to key, we can also fetch documents by their content
  - can be hierarchical – a value can be a JSON object or a list
- Instead of relations/tables we have collections
  - a set of documents
  - common structure among documents in the same collection is NOT enforced
- Use Javascript instead of SQL to interact
- Data is in JSON
- https://docs.mongodb.com/manual/reference/sql-comparison/

JSON

- JavaScript Object notation
- {Key: Value}
  - var data = {“Name”: “Alice”, “Major”: CS”, “University”: “UofM”, “Hobby”: “Beating MSU”};
  - Can have nested key values as well:
    - {“Location”: {“City”: “Ann Arbor”, “State”: “Michigan”, “Country”: “USA”}};
- Retrieve data by data[“Name”]
  - Returns “Alice”
- JSON Objects are not ordered



note: JSON has nothing to do with Jason Momoa, but again, they sound similar

SQL vs. MongoDB

SQL	MongoDB
Tuple	Document. Represented as a JSON object
Relation/Table	Collection. Represented as a JSON array
`SELECT * FROM Users;`	`db.users.find();`

`SELECT * FROM Users`
 	
`WHERE name = 'John' AND age = 50;`
	`db.users.find({name: 'John', age: 50});`

`SELECT user_id, addr FROM Users`
 	
`WHERE name = 'John';`
	`db.users.find({name: 'John'},`
	
	`{user_id: 1, addr: 1, _id: 0});`

Project 3 Intro

Project 3

- Part A: Java code to export database to JSON
  - Need to perform this on CAEN just like in Project 2
  - Extract data from tables in the Fakebook database and exporting a JSON file
  - Does not use MongoDB

Project 3

- Part B: MongoDB Queries
  - You can run entirely on your local machine (requires installing MongoDB)
    - We won’t be able to help with specific installation issues though
  - Use the eecs484 server through CAEN
    - Set up your project on CAEN and edit the Makefile as specified in the spec
    - Run commands which will connect to a MongoDB server setup for you on the eecs484 server
  - Write the queries!
    - Lots of helpful references in the spec to various MongoDB documentation

General Tips

- Try to focus on getting the solution correct rather than doing it in the fanciest way
  - Yes you can use an aggregate, group, out pipeline or you can iterate a couple times
  - No efficiency tests
  - No private tests
  - Most important thing: make sure you understand how and why your code works
- Take the time to get familiar with Javascript
  - Documentation will be your best friend
- Query 5 is the hardest
  - Has a similar concept to how you dealt with the friends relation in Query 6 on Project 2
    - Completely different code but similar work around needed

Get started on Project 2 and 3!

We’re here if you need any help!!

- Office Hours: Schedule is here, both virtual and in person offered
- Piazza
- Next week’s discussion!!!