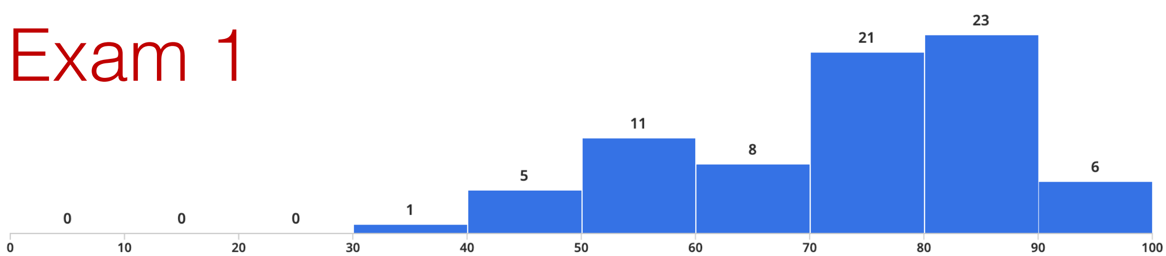
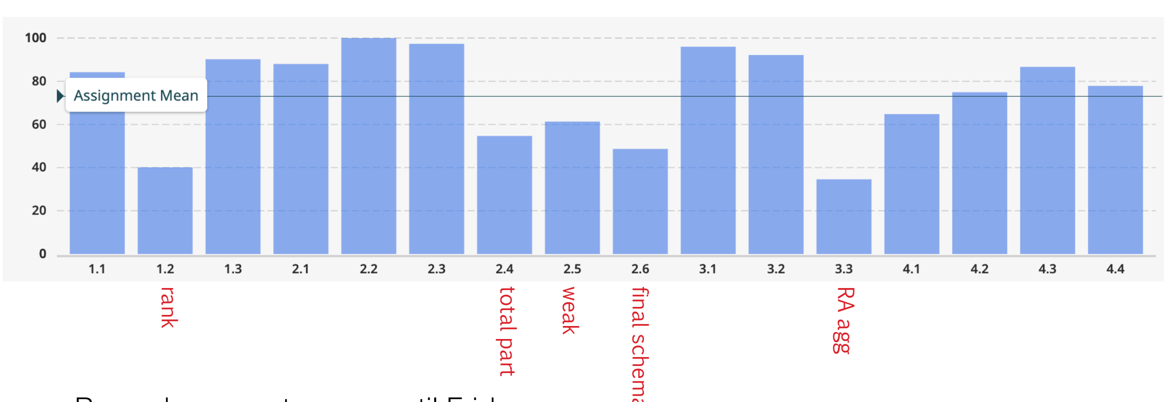
Outline

- Storage hierarchy
- Disks
- RAID
- Spark
- Buffer Manager
- File Organization
- Indexes
- B+-Tree Indexes
- Etc..

Exam 1



Minimum	Median	Maximum	Mean	Std Dev
36.0	75.0	94.5	72.71	14.25

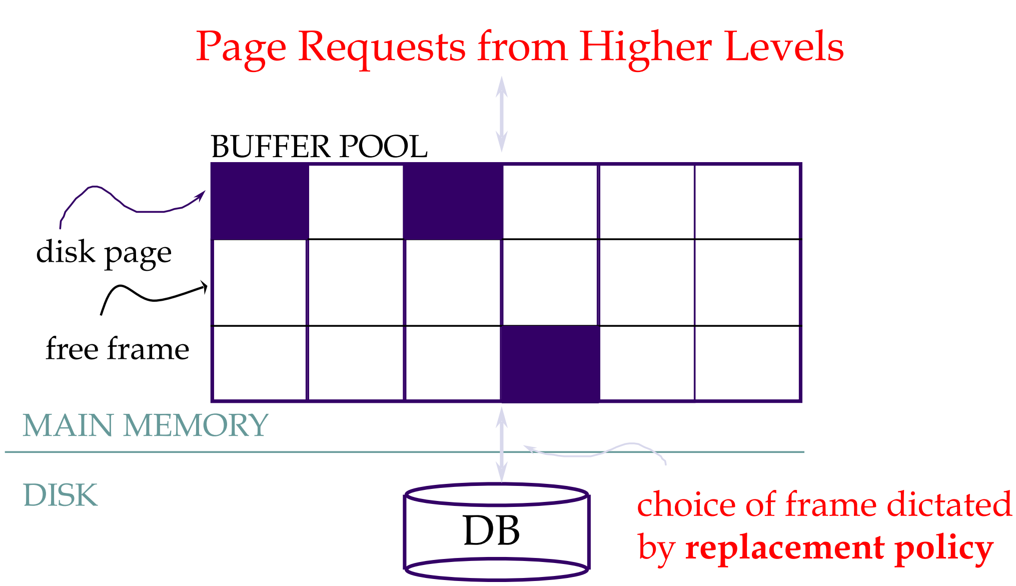


- Regrade requests open until Friday noon.

Buffer Manager

- When the QP wants a block, it asks the “buffer manager”
  - The block must be in memory to operate upon
- Buffer manager:
  - If block already in memory: return a pointer to it
  - If not:
    - Evict a current page
      - Either write it to temporary storage,
      - or write it back to its original location,
      - or just throw it away (if it was read from disk, and not modified)
    - and make a request to the storage subsystem to fetch it

Buffer Manager



Buffer Manager

a b c d e f d e f

- Similar to *virtual memory manager*
- Buffer replacement policies
  - Which page to evict ?
  - LRU: Least Recently Used
    - Throw out the page that was not used in a long time
  - MRU: Most Recently Used
    - The opposite
    - If data set too big for cache, keep older pages as they might be accessed again before recent pages.
  - Clock ?
    - An efficient implementation of LRU

Buffer Manager

- LRU: Least Recently Used

input	A	B	A	B	C	B	A	F
buffers	A	B	A	B	C	B	A	F
		A	B	A	B	C	B	A
	miss	miss			miss		miss	miss

- But LRU can be bad, such as when looping over array bigger than space:

MRU

input	A	B	C	D	A	B	C	D	A	B
buffers	A	A	A	A	A	A	A	A	A	A
		B	B	B	B	B	B	B	B	B
			C	D	D	D	C	D	D	D
	miss	miss	miss	miss			miss	miss		

LRU

input	A	B	C	D	A	B	C	D	A	B
buffers	A	B	C	D	A	B	C	D	A	B
		A	B	C	D	A	B	C	D	A
			A	B	C	D	A	B	C	D
	miss	miss	miss	miss	miss	miss	miss	miss	miss	miss

DB Needs from Buffer Manager

- *Pinning* a block
  - Not allowed to evict
- *Force-output (force-write)*
  - Force the contents of a block to be written to disk
- *Order the writes*
  - *This* block must be written to disk before *that* block

Critical for fault tolerant guarantees

- Otherwise database has no control over what is on disk

Outline

- Storage hierarchy
- Disks
- RAID
- Spark
- Buffer Manager
- File Organization
- Indexes
- B+-Tree Indexes
- Etc..

File Organization

- How are the relations mapped to the disk blocks ?
  - Use a standard file system ?
    - High-end systems have their own OS/file systems
    - OS interferes more than helps in many cases
  - Mapping of relations to file ?
    - One-to-one ?
    - Advantages in storing multiple relations clustered together
  - A *file* is essentially a *collection of disk blocks*
    - How are the tuples mapped to the disk blocks ?
    - How are they stored within blocks?

File Organization

- Goals:
  - Allow insertion/deletions of tuples/records
  - Fetch a particular record (specified by record id)
  - Find all tuples that match a condition (say SSN = 123) ?
- Simplest case
  - Each relation is mapped to a file
  - A file contains a sequence of records
  - Each record corresponds to a logical tuple
- So….
  - How are tuples/records stored within a block ?

Fixed Length Records

- n = number of bytes per record
- Store record *i* at position: $n * (i - 1)$
- Records may cross blocks
  - Not desirable
  - Stagger so that that doesn't happen

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

- Inserting a tuple ?
  - Depends on the policy used
  - One option: Simply append at the end of the file. Problems?
- Deletions ?
  - Option 1: Rearrange
  - Option 2: Keep a *free list* and use for next insert

The above assumes records not ordered.

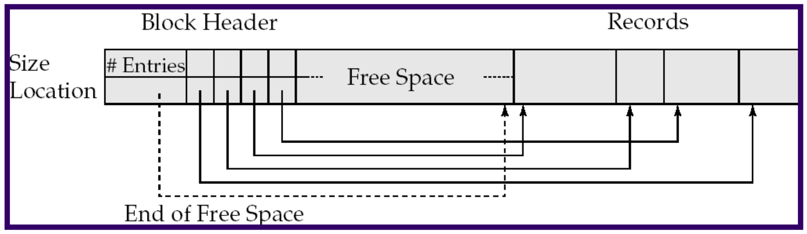
Fixed Length Records

- Deleting: using "free lists"

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Variable-length Records

Slotted page structure



- *Indirection:*
  - The records may move inside the page, but the outside world is oblivious to it
  - Why ?
    - The headers are used as an indirection mechanism
    - "Record ID 1000 is the 5th entry in page X"

File Organization

- Which block of a file should a record go to ?
  - Anywhere ?
    - Called "heap" organization
    - How to search for "SSN = 123" ?
  - Sorted by SSN ?
    - Called "sequential" organization
    - Keeping it sorted might be painful
    - How would you search ?
  - Based on a hash key
    - Called "hashing" organization
    - Store the record with SSN = x in the block number h(x)
    - Why ?

Sequential File Organization

- Keep sorted by some *search key*
- Insertion
  - Find the block in which the tuple should be
  - If there is free space, insert it
  - Otherwise, must create overflow pages
- Deletions
  - Delete and keep the free space
  - Databases tend to be insert heavy, so free space gets used fast
- Can become *fragmented*
  - Must reorganize once in a while

Sequential File Organization

- What if I want to find a particular record by value ?
  - *Account info for SSN = 123*
- Binary search
  - Takes $ceiling(log_2(n))$ number of disk accesses
    - These are *random* accesses
  - Too much
    - n = 1,000,000,000, $log_2(n) = 30$
    - Assume each random access approx 5 ms
    - 150 ms to find just one account information
    - < 7 requests satisfied per second

Hash-based File Organization

(as opposed to sequential file organization)

Store record with search key k
in block number h(k)

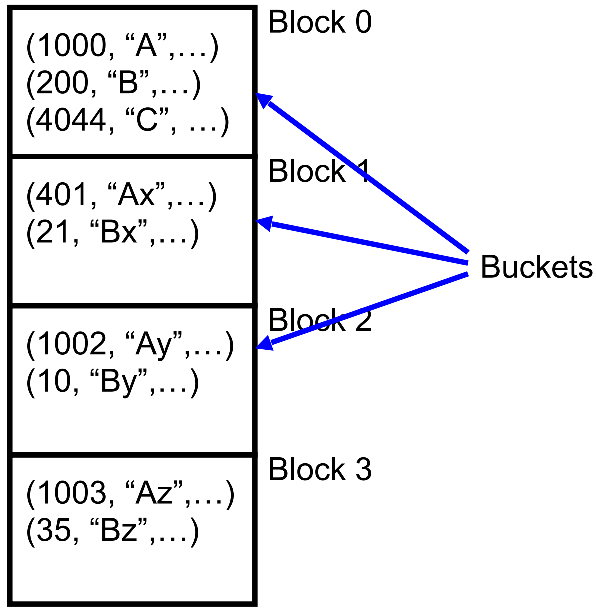
e.g. for a person file,
h(SSN) = SSN % 4

Blocks are the buckets

What if the block becomes full ?
Overflow pages

Uniformity property:
Don't want all tuples to map to the same bucket
h(SSN) = SSN % 2 would be bad

No index needed
No reasonable range queries



Hash-based File Organization

Hashed on "branch-name"

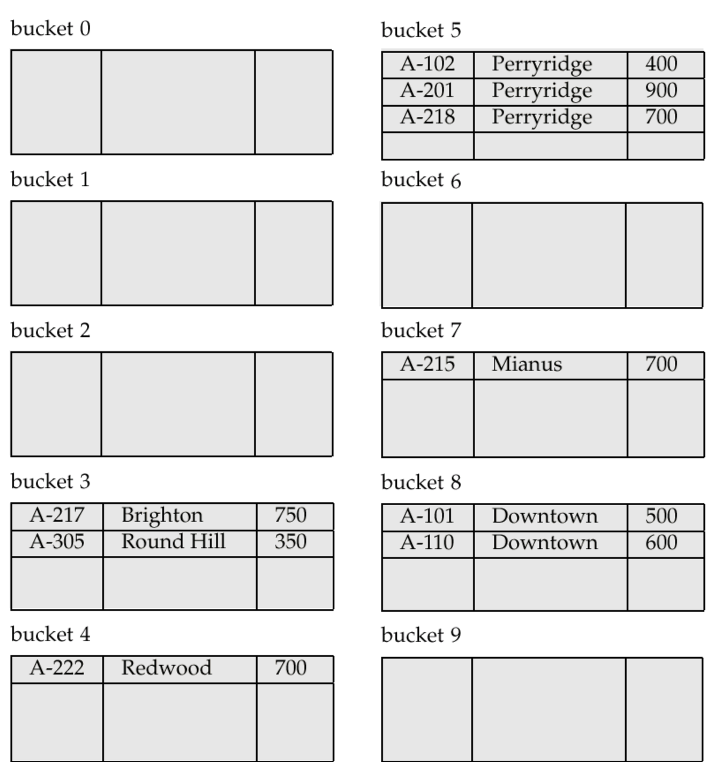
Hash function:

a = 1, b = 2, .., z = 26

h(abz)

= (1 + 2 + 26) % 10

= 9



Outline

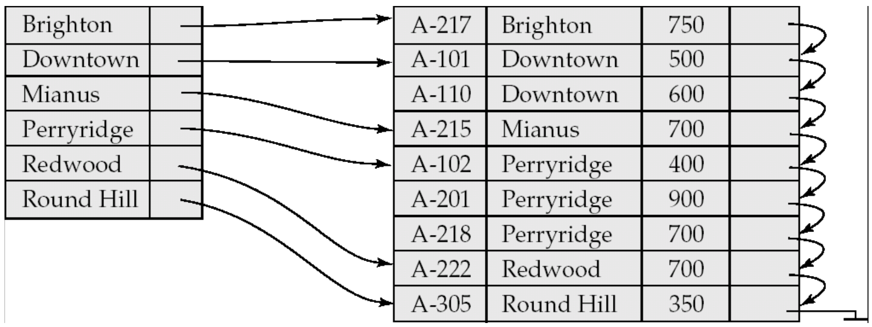
- Storage hierarchy
- Disks
- RAID
- Spark
- Buffer Manager
- File Organization
- Indexes
- B+-Tree Indexes
- Etc..

Index

- A data structure for efficient search through large databases
- Two key ideas:
  - The records are mapped to the disk blocks in specific ways
    - Sorted, or hash-based
  - Auxiliary data structures are maintained that allow quick search
    - Think library index/catalogue
- *Search key*:
  - Attribute or set of attributes used to look up records in indexes
  - E.g. SSN for a persons table
  - *Can be different from candidate or primary keys*
- Two main types of indexes
  - Ordered indexes
  - Hash-based indexes

Ordered Indexes

- We assume ordered indexes are sorted by search key
- Primary (“clustered”) indexes
  - File ordering = search key
  - Can have only one primary index on a relation
- Secondary (“nonclustered) index
  - File ordering != search key



- *dense* means every search value has an index entry