Review of Last Lecture

- Transaction
  - ACID
  - States
- Transaction in SQL
- 4 Isolation Levels in SQL

Today's Plan

- How data can be stored in database
  - Unordered file
  - Ordered file
  - Hashing
  - B+ Tree
- Indexing

STORAGE AND INDEXING

CHAPTER 10

Collin Roberts

University of Waterloo

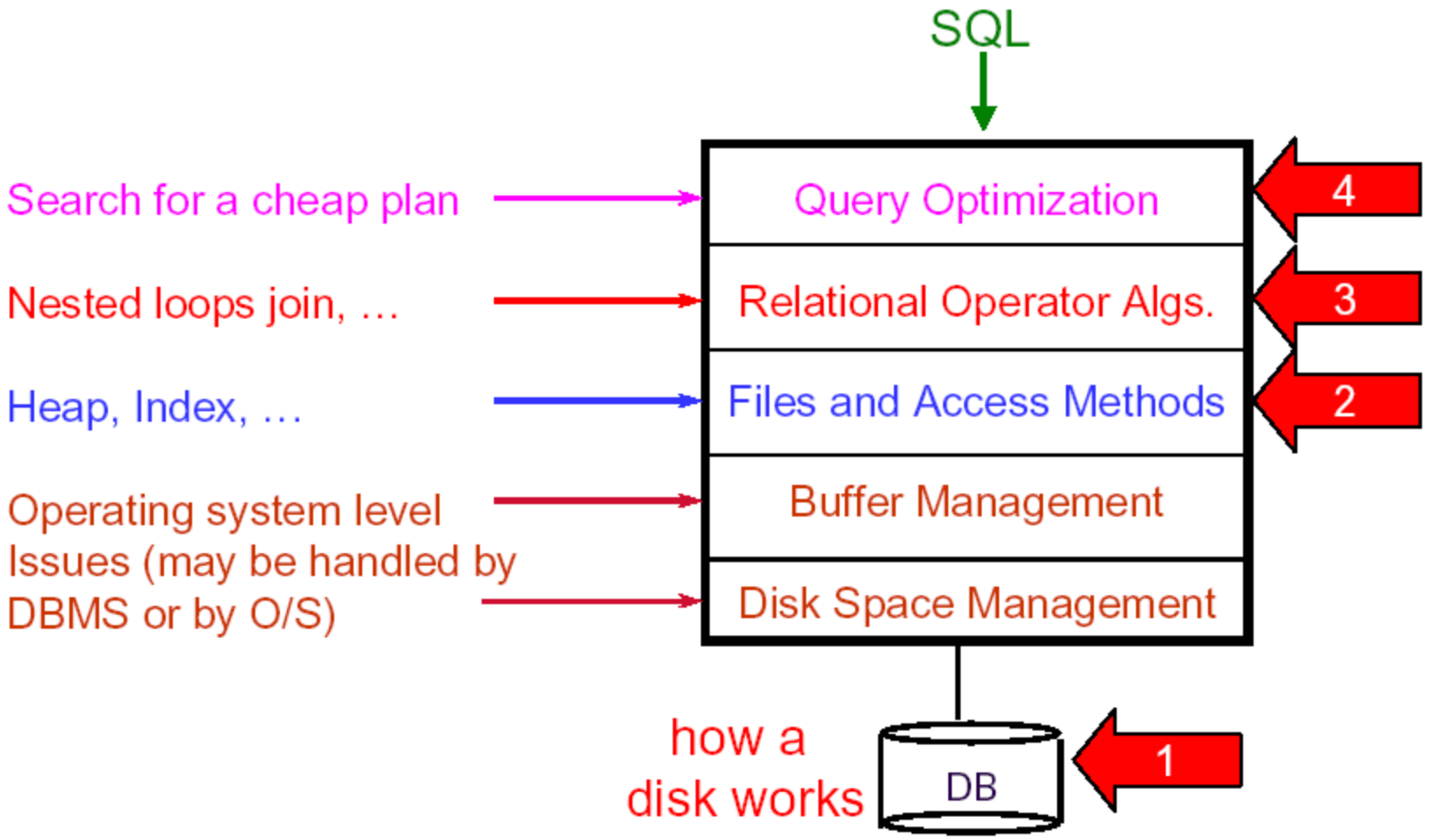
Recommended Readings

- Textbook Chapters 13 and 14

Why are we interested in Storage and Indexing?

- Database systems already do this…
- But it is important to understand how they work, and
- the techniques are also useful for other applications!
- Google's Linux cluster currently processes over *150 million queries per day*, *searching a multi-terabyte web index for every query with an average response time of less than a quarter of a second* with near-100% uptime.
- Traditional relational databases are not suitable for Google
- Knowledge of database techniques required to build custom solutions
  - How to store data? Goal: efficiency, high-availability
  - How to efficiently access data? Goal: low response time

DBMS Structure



How to access data faster?

- Come up with a better way to execute the query
  - Query optimization, already discussed (briefly) in Chapter 03
- Come up with a better way to logically organize the data
  - Data structure of database and indexing
- Come up with a better way to physically store the data on hard drive
  - Storage technology e.g. RAID

How to access data faster?

- Come up with a better way to answer the query
  - Query optimization, already discussed in Chapter 03
- Come up with a better way to logically organize the data
  - Data structure of database and indexing
- Come up with a better way to physically store the data on hard drive
  - Storage technology and RAID

How to store and organize the data logically in a table?

- Ontario has about 650,000 public employees. How many of them are in the sunshine list (salary $100K+)?

$$\sigma_{\text{Salary>=100,000}}(\text{Employee})$$

- What if the employee records are randomly stored in the table? How many rows needs to be accessed?

FName	LName	Sex	Dept	Rank	…	…	Address	SIN	Salary
John	Doe	M	Acct	L3	…	…	Toronto	111145	55,000
Jane	Jane	F	Tranp	L2	…	…	Waterloo	234435	45,000
Kate	Lo	F	IT	L10	…	…	Guelph	343999	110,00
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Unordered Files

- Rows are not sorted, also called a **heap** or a **pile file**.
- New records are inserted at the end of the file. Hence, record insertion is quite efficient.
- To search for a record, a **linear search** through the file records is necessary. This requires reading and searching half the file blocks on the average, and is hence quite expensive.
- Worse yet: reading the records in order of a particular field requires sorting all file records.

Ordered/Sorted Data: Motivation

- Ontario has about 650,000 public employees. How many of them are in the sunshine list (salary $100K+)?

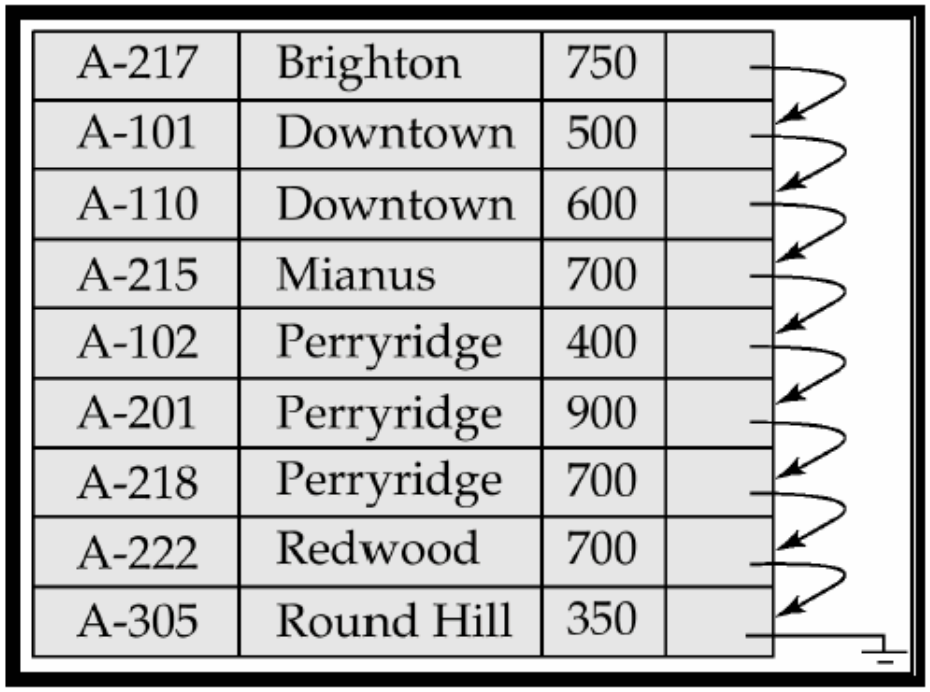
$$\sigma_{\text{Salary>=100,000}}(\text{Employee})$$

- What if the employee records are sorted based on salary in the table? How many rows needs to be accessed?

FName	LName	Sex	Dept	Rank	…	…	Address	SIN	Salary
Jane	Jane	F	Tranp	L2	…	…	Waterloo	234435	45,000
John	Doe	M	Acct	L3	…	…	Toronto	111145	55,000
Kate	Lo	F	IT	L10	…	…	Guelph	343999	110,00
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Ordered File Organization: Idea

- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key



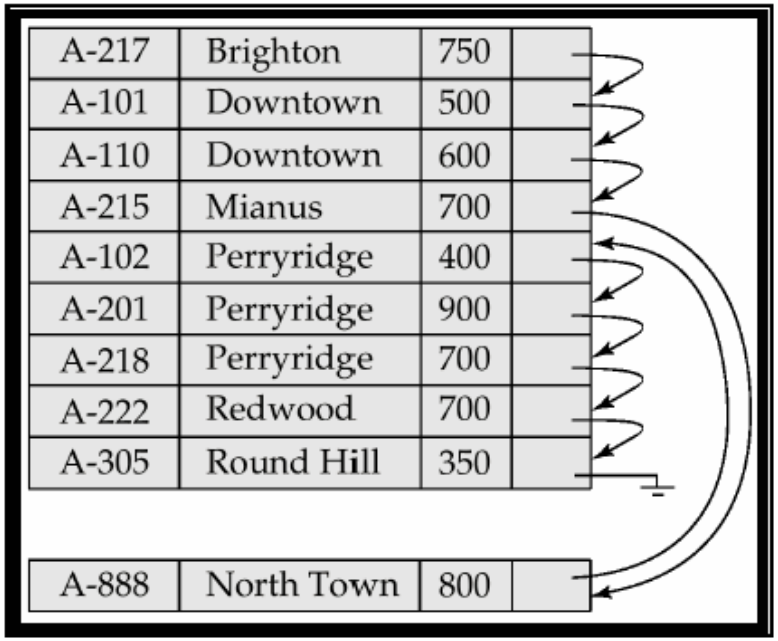
A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

Example: Performance Discrepancy

- In a list of one million integers, check whether 123,456 is in the list. Say one row stores one integer.
- Number of rows accessed: unordered file vs ordered file?
  - Sequential search for unsorted list
  - Binary search for sorted list: https://en.wikipedia.org/wiki/Binary_search_algorithm

Ordered File Organization: Problem

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



Hash Files (FYI)

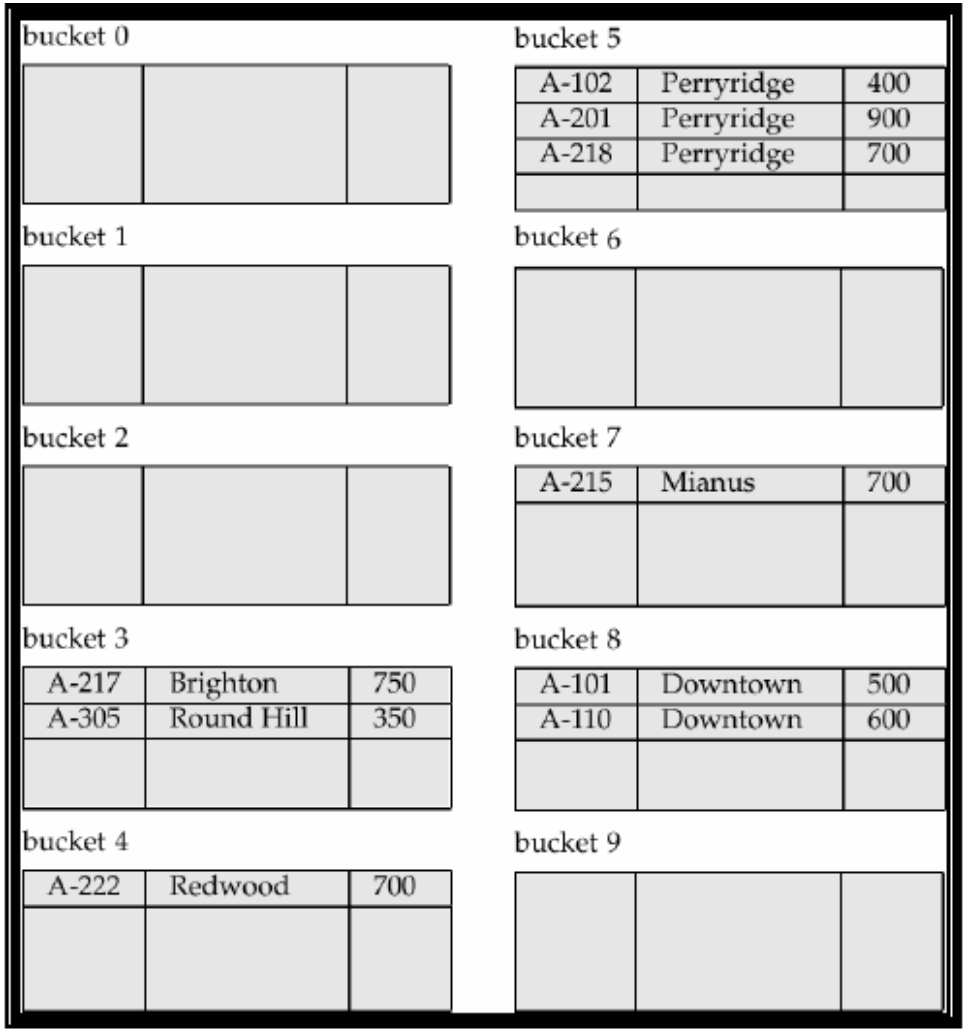
- The file is divided into M equal-sized buckets, numbered bucket0, bucket1, ..., bucket M-1. Typically, a bucket corresponds to one (or a fixed positive number of) disk blocks.
- A hash function h determines item K is stored in bucket i, where i=h(K)
  - Search is very efficient on K.
  - Inserting K is also very efficient.
- Collisions occur when a new record hashes to a bucket that is already full.
  - An overflow file is kept for storing such records. Overflow records that hash to each bucket can be linked together.

Hash File: Example I (FYI)

Hash file organization of *account* file, using *branch-name* as key
(See figure in next slide.)

- There are 10 buckets,
- The binary representation of the *i*th character is assumed to be the integer *i.*
- The hash function returns the sum of the binary representations of the characters modulo 10
  - E.g. h(Perryridge) = 5 h(Round Hill) = 3
    h(Brighton) = 3

Hash File: Example I (Continued)



B^+-tree File Organization (FYI)

- Index file degradation problem is solved by using B^+-Tree indices. Data file degradation problem is solved by using B^+-Tree File Organization.
- The leaf nodes in a B^+-tree file organization store records, instead of pointers.
- Since records are larger than pointers, the maximum number of records that can be stored in a leaf node is less than the number of pointers in a nonleaf node.
- Leaf nodes are still required to be half full.
- Insertion and deletion are handled in the same way as insertion and deletion of entries in a B^+-tree index.

Index: Motivation

- Sorted list significantly improve access performance for search
  - Note that data stored in ordered file, hash table or B^+-tree is all sorted in some way.
- But file/data can only be stored physically in one order!
  - Say sorted by customer names. Then search by name can be done using binary search
  - But what if we want to search by age?

Example: Search by Name vs. by Age

Indexes as Access Paths

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.

- The index is usually specified on one attribute of the file (although it could be specified on several attributes)

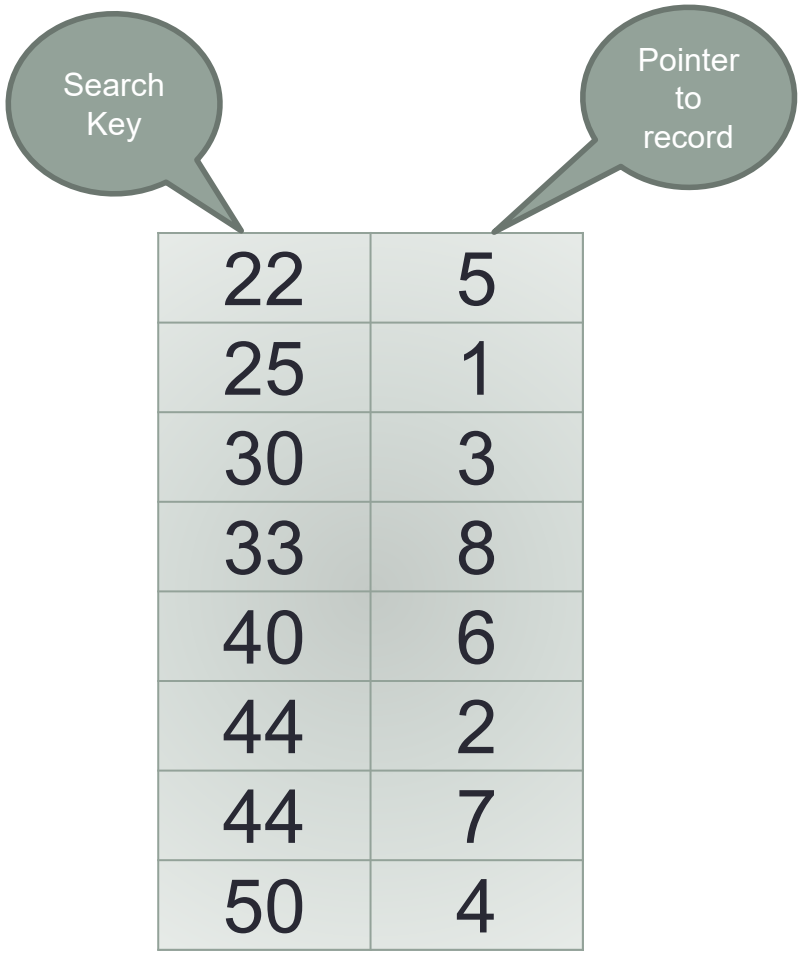
- One form of an index is a file of entries

  <search key, pointer to record>,

  which is ordered by the search key (attribute value).

  So search on the search key is quite efficient

Index Example: Age



Ashby, 25, 3000

Basu, 44, 4003

Bristow, 30, 2007

Cass, 50, 5004

Daniels, 22, 6003

Jones, 40, 6003

Smith, 44, 3000

Tracy, 33, 5004

How Index Help: Example (FYI)

Example: Given the following data file:
EMPLOYEE(NAME, SSN, ADDRESS, JOB, SAL, ...)
Suppose that:
record size R=150 bytes
block size B=512 bytes
r=30000 records

Then, we get:
blocking factor Bfr= B div R= 512 div 150= 3 records/block
number of file blocks b= (r/Bfr)= (30000/3)= 10000 blocks

For an index on the SSN field, assume the field size V_{SSN}=9 bytes, assume the record pointer size P_R=7 bytes. Then:
index entry size R_I=(V_{SSN}+ P_R)=(9+7)=16 bytes
index blocking factor Bfr_I= B div R_I= 512 div 16= 32 entries/block
number of index blocks b= (r/ Bfr_I)= (30000/32)= 938 blocks
binary search needs $\log_2$bI= log2938= 10 block accesses

This is compared to an average linear search cost of:
(b/2)= 30000/2= 15000 block accesses
If the file records are ordered, the binary search cost would be:
$\log_2$b= $\log_2$30000= 15 block accesses

Index in SQL

- Create an index

  create index <index-name> **on** <relation-name> (<attribute-list>)

  E.g.: **create index** *b-index* **on** *branch(branch-name)*

- Use **create unique index** to indirectly specify and enforce the condition that the search key is a candidate key
  - Not really required if SQL **unique** integrity constraint is supported
- To drop an index

  drop index <index-name>

Summery of Today's Lecture

- How data can be stored in database
  - Unordered file
  - Ordered file
  - Hashing
  - B+ Tree
- Indexing