Lecture 8: File Organization

Last few days: File systems.

Today: File Organization

- Serial files
- Queries
- Updates
- Batch Update
- Co-Sequential Processing.

Folk & Zoellick, § 4.1, 7.1, 7.2

Levels of Data Access

- A file system allows a user to read/write bytes into a file.

- eg. Retrieve the 130th byte of file F, or retrieve bytes 1,500 to 1,700.

- However, users do not usually know which bytes hold the data they are interested in.

- eg. A nurse may request data on patient "Woody Allen". This data may be stored in bytes 100,330 — 100,597, but the nurse does not know this.

- Files must be <u>organized</u> so that data can be (quickly) accessed in human terms

- eg. Given the name of a patient, quickly Find his record in a file.

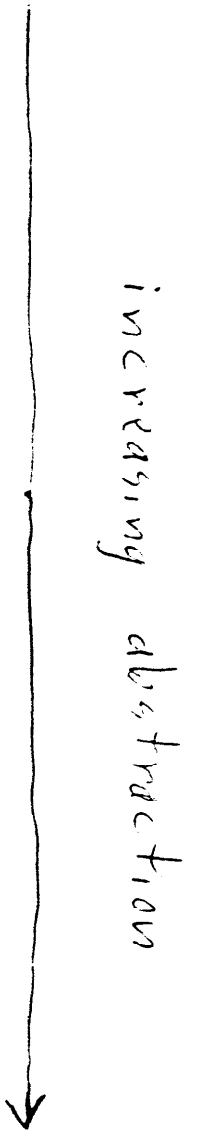
- ie, must convert a patient's name into a byte number in a file.

Levels of Data Access / Abstraction

Disk: Retrieve the block of bytes at cylinder i, surface j, and track k.

File System: Retrieve bytes i through j of File F.

Database System: Retrieve Woody Allen's medical records



File Organization

Four basic types of organization:

Much of this course.

- Serial ← today
- Direct
- Indexed
- Multi-Key

In all cases we view a file as a sequence of records.

- A record is a list of fields.

- eg. An employee file is a sequence of employee records.

- An employee record may have fields such as

  - name
  - address
  - employee number
  - salary
  - etc.

- Each field has a datatype.

  eg, string, integer, floating point, etc.

- A field that uniquely identifies a record is called a <u>key</u>.

  eg. employee number, student number, ...

File Operations

- Typical Operations:
  - Insert a record
  - Delete a record
  - Modify a field of a record
  - Retrieve a record.

- In Direct Files:

  Get a record with a given field value.

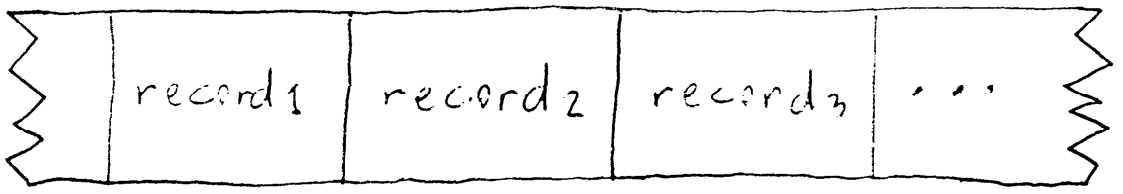
- In Serial Files:

  Get the next record.

Serial Files

- Records are stored contiguously on a storage device.

- eg.



- Typical of files stored on tape.

- If the records are stored in sorted order (sorted by some field), then the file is called sequential.

Searching Serial Files

- Typical Request:

  Print all records with a given field value.

  eg. Print all courses taught by Jones.

- If the file is unsorted, we must examine each record in the file, in order, starting from the first record.

- Requires N+1 record accesses.

  ↑ 1 extra access for the end-of-file (eof) marker.

  Here, N is the number of records in the file.

Example 1: Unsorted, serial access.

3 Fields

Record #	Prof.	Course
1	Jones	CSC 158
2	Bonner	CSC 228
3	Smith	CSC 434
4	Bonner	CSC 324
5	Jones	CSC 459
6	Smith	CSC 110
7	Smith	CSC 260
8	eof	

7+1 records

Sample Queries

① Print all courses taught by Jones.

- 8 records are read.
- 2 records are printed.

② Print all courses taught by Bonner.

- 8 records are read
- 2 records are printed.

③ Print all courses taught by Smith

- 8 records are read.
- 3 records are printed.

Note: In all cases, N+1 records are read (ie, 7+1 records).

Example 2: Sorted, serial access.

Record #	Prof	Course
1	Bonner	CSC 228
2	Bonner	CSC 324
3	Jones	CSC 158
4	Jones	CSC 439
5	Smith	CSC 434
6	Smith	CSC 110
7	Smith	CSC 260
8	eof	

Sorted by Prof

Sample Query: Print all courses taught by Jones

- Starting with the first record, examine each record in turn, until the value of the key field (Prof) equals the given value (Jones)

- In this case, 5 records are read.

In general, such a query requires

- at least 1 record access
- at most N+1 record accesses.

N+1: all records + plus eof.

$\therefore \frac{1 + (N+1)}{2} = \frac{N}{2} + 1$ accesses on average

$\frac{N}{2}$: half the records

+1: we always get 1 record too far.

Examples

① Print all courses taught by Jones.

- Requires 5 record accesses

② Print all courses taught by Bonner.

- Requires 3 record accesses.

③ Print all courses taught by Smith.

- Requires 8 record accesses.

Average number of accesses

$$= \frac{5+3+8}{3} = \frac{16}{3} = 5\frac{1}{3} \approx \frac{7}{2} + 1 = 4\frac{1}{2}$$

Updating Serial Files

- Updates to a serial file are expensive, since each update requires reading half the file (on average).

- eg. Change Fred's salary to $50,000.
  We must read the file sequentially until Fred's record is found.
  Then, we change the salary field of this record to 50,000.

- In general, M updates take O(M·N) record accesses (on average).

- This is expensive,

Batched Updates

- Sometimes, changes to a file must be made immediately (eg. air flight reservations).

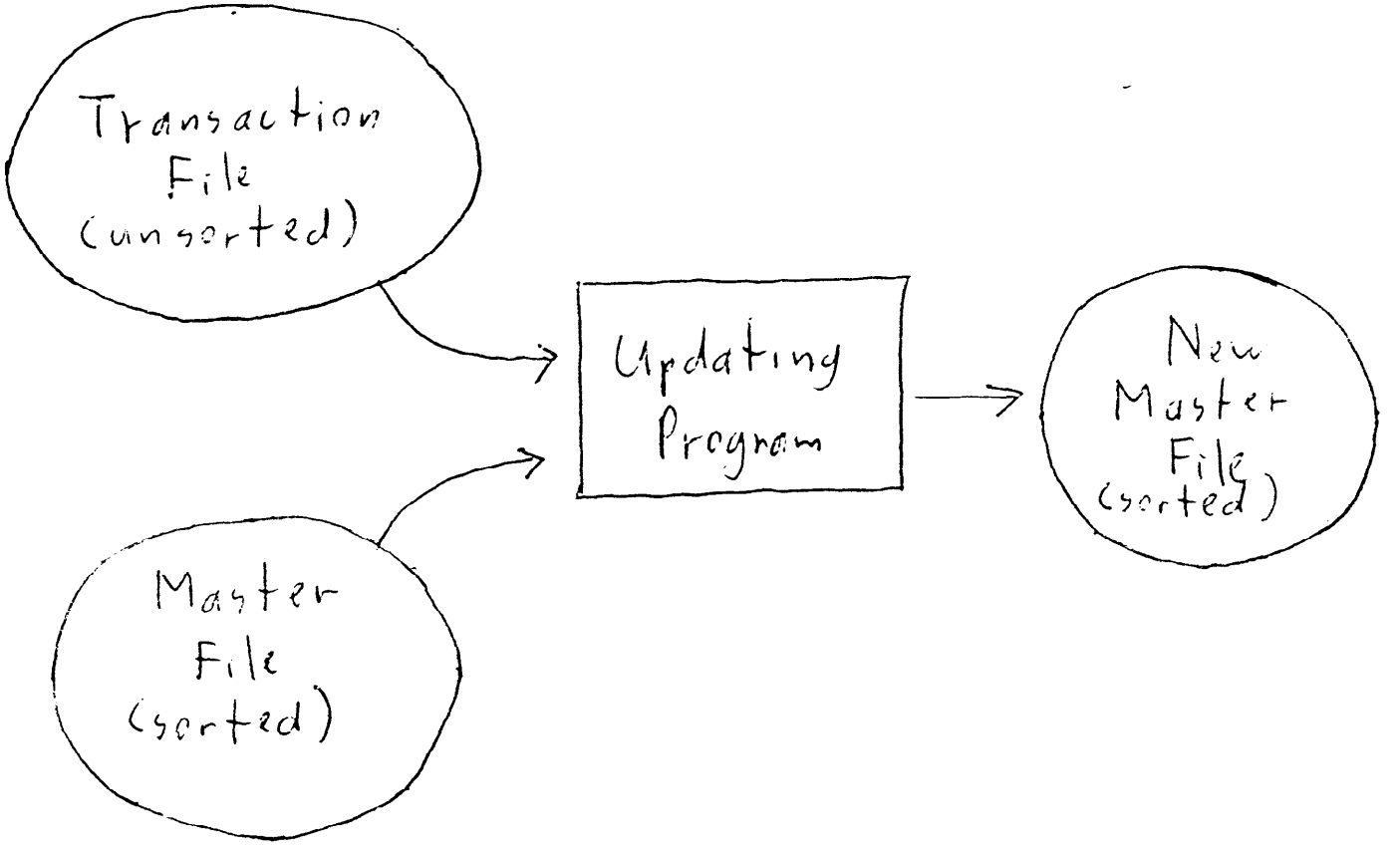
- In such cases serial Files are too slow to be practical

- Other times changes can be <u>batched</u>, ie, recorded on a transaction File and processed later (eg student registration)

- In such cases, cost can be reduced from $O(N \cdot M)$ record access to $O(N + M \log M)$ record accesses, as we shall see.

Co-Sequential Processing

Example: Batch Update.



Master file is sorted by key.

M = # Transactions

N = # Master-File records.

Step ①:

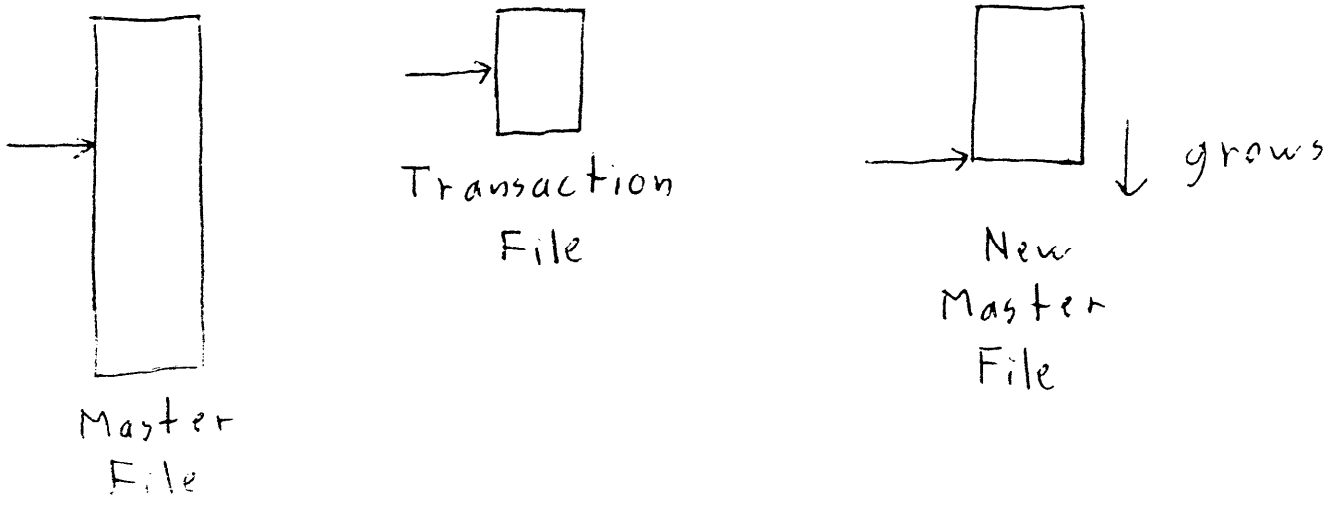
Sort the transaction file by key.

Requires $O(M \cdot \log M)$ file accesses.

Step ②:

Scan the (sorted) Master file and the (sorted) Transaction file simultaneously while generating a new master file.

Requires $O(M+N)$ file accesses.



Example

Master File

Bach
Beethoven
Chopin
Greig
Mendelssohn
Mozart
Rachmaninoff
Ravel
Tchaikovsky
Vivaldi

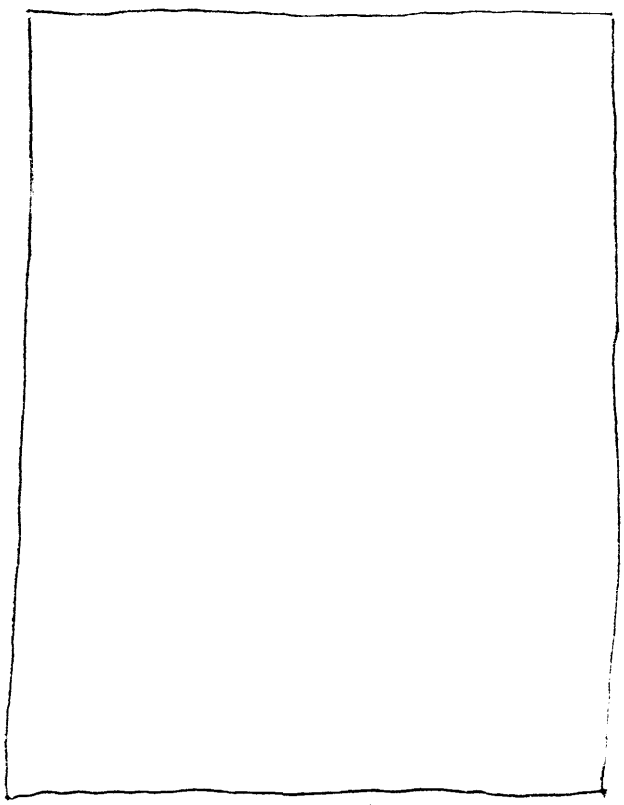
(sorted)

Transaction File

insert: Eisner
delete: Mozart
modify: Ravel

(sorted)

New Master File



(sorted)

Algorithm for Co-Sequential Batch Update

First, initialize pointers to the first records in the master and transaction files.

Second, do until pointers reach eof:

- Compare the keys of the current records in the Master and Transaction Files.

- Take appropriate action.

- advance one (or both) of the pointers.

Appropriate Action

If master key < transaction key

then Copy master file record to the end of the new master file.

Advance the master file pointer.

If master key > transaction key

then If transaction is an insert

then Copy transaction file record to the end of the new master file.

else Log an error

Advance the transaction file pointer

(Because the record referred to in the transaction file is not in the master file)

- If master key = transaction key, then
  - If transaction is <u>modify</u>, then
    Copy modified master file record to the end of the new master file.
  - If transaction is <u>insert</u>, then
    Log an error (since the record to be inserted is already in the master file).
  - If transaction is <u>delete</u>, then
    Do nothing.

In all three cases, advance both the master and transaction file pointers.

(<u>Note:</u> If the transaction file contains multiple records with the same key, special care is needed

Cost

N = # Master File records

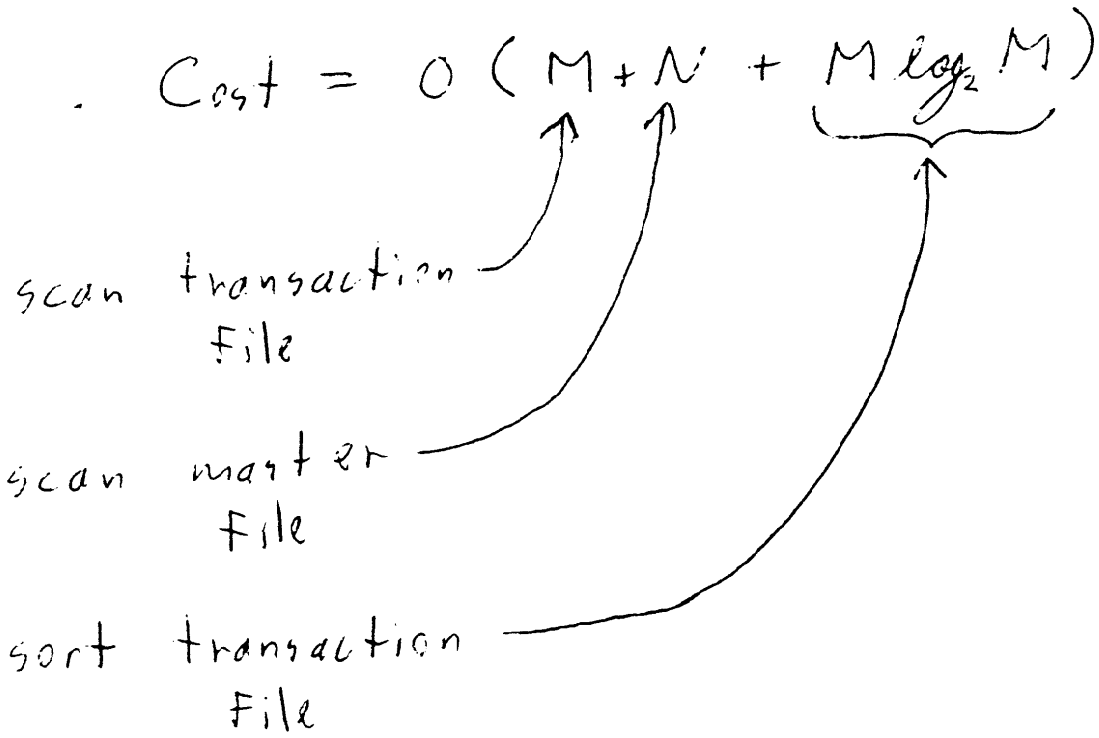
M = # Transaction file records

Immediate Update:

$$\text{Cost} = O(M \cdot N)$$

Batch Update:

$$\text{Cost} = O(M + N + M \log_2 M)$$



Example: Library Database

N = # books in library = 1,000,000.

M = # books borrowed or returned each day

= 2,000.

1) Suppose each update is processed immediately.

Cost $\approx M \cdot N$ = 2,000,000,000 File accesses.

2) Suppose we record each update on a transaction file, and update the master file in batch at night.

Cost $\approx M + N + M \log_2 M$

$= 2{,}000 + 1{,}000{,}000 + 2000 \cdot \log_2 2000$

$\leq 1{,}002{,}000 + 2000 \cdot 11$

$= 1{,}024{,}000 \ll 2{,}000{,}000{,}000$

Summary of Serial Files

Advantages:

1. Fast access to next record in sequence.
2. Cheap storage media (Tape).
3. Easy to do file back up.

Disadvantages:

1. Retrieval is sequential.
2. Updates can be slow.
3. Updates usually need to be batched.