Chapter 6 Organizing Files for Performance.

Reclaiming Lost Space

Finding Things Quickly

RECLAIMING LOST SPACE:

Modifications to records cause deterioration of the file organization

Modifications

- Add a record
- Update a record
- Delete a record

Fixed Length Records

Deletion is the only problem

Variable Length Records

Deletion and Update cause problems

Adds that attempt to reclaim deleted space cause problems

Update is viewed as delete old followed by add changed record

DELETION AND COMPACTION

- After we delete a number of records we must recover the space used by these records
- This recovery is called compaction
- In fact compaction is defined as recovering space not currently used by data.

Example: File:

Zaphod|Beeblebrox|1800 Star|Betaljuice|KS|69432| …

Ford|Perfect|2700 Sun|Gillford|UK|GF444|…

Arthur|Dent|100 Earth|Gillford|UK|GF111|…

delete Ford Perfect, indicate by marking the record

Zaphod|Beeblebrox|1800 Star|Betaljuice|KS|69432| …

*|Perfect|2700 Sun|Gillford|UK|GF444|…

Arthur|Dent|100 Earth|Gillford|UK|GF111|…

After compaction the record is gone

Zaphod|Beeblebrox|1800 Star|Betaljuice|KS|69432| …

Arthur|Dent|100 Earth|Gillford|UK|GF111|…

Compaction programs are run on a schedule that is determined by the volatility of the data.

DYNAMIC RECLAMATION OF SPACE

For files that are highly volatile and have a lot of records it is not practical to use the above mentioned scheme

We need to use the space "immediately". So:

1. We need to mark the records as deleted
2. We need to know if there are empty slots for new records
3. We need a way to locate these slots when we need one

Fixed Length Records – This is the simplest

- Create a stack of available records
- Store this stack in the available space
- Put the location of the first unused record in the header record

When the file is created		
	USED	UNUSED
	0	1
1	*2	
2	*3	
3	*4	
4	*5	
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Add Zaphod to the file		
	USED	UNUSED
	1	2
1	Zaphod …	0
2	*3	
3	*4	
4	*5	
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Add Trillium		
	USED	UNUSED
	2	3
1	Zaphod	0
2	Trillium	1
3	*4	
4	*5	
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Add Ford		
	USED	UNUSED
	3	4
1	Zaphod	0
2	Trillium	1
3	Ford	2
4	*5	
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Add Marvin		
	USED	UNUSED
	4	5
1	Zaphod	0
2	Trillium	1
3	Ford	2
4	Marvin	3
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Delete Trillium		
	USED	UNUSED
	4	2
1	Zaphod	0
2	*5	
3	Ford	1
4	Marvin	3
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Delete Marvin		
	USED	UNUSED
	3	4
1	Zaphod	0
2	*5	
3	Ford	1
4	*2	
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

Add Arthur		
	USED	UNUSED
	4	2
1	Zaphod	0
2	*5	
3	Ford	1
4	Arthur	3
5	*6	
6	*7	
7	*8	
8	*9	
9	*10	
10	*11	
11	*12	
12	*0	

When asked to list objects in the file. You must look at the contents because that is how you find out if it is used or not.

DELETING VARIABLE LENGTH RECORDS

Use available list of variable length records.

Use byte counts instead of relative record numbers

Empty File

Used	Unused	
-1	0	

Add Zaphod

Used	Unused		
0	8	08Zaphod	
		0	8

Add Trillium

Used	Unused			
0	18	08Zaphod	10Trillium	
		0	8	18

Add Ford

Used	Unused				
0	24	08Zaphod	10Trillium	06Ford	
		0	8	18	24

Add Marvin

Used	Unused					
0	32	08Zaphod	10Trillium	06Ford	08Marvin	
		0	8	18	24	32

Delete Trillium

Used	Unused					
0	8	08Zaphod	10*32	06Ford	06Marvin	
		0	8	18	24	32

Delete Ford

Used	Unused					
0	18	08Zaphod	10*32	06*08	06Marvin	
		0	8	18	24	32

Add Arthur

Used Unused

0	18	08Zaphod	10Arthur//	06*32	08Marvin	
		0	8	18	24	32

- The spot vacated by Ford was and is too small for Arthur so check the next empty spot.
- Arthur fits so put him in.
- Since Arthur doesn't use all the space we just ignore the unused space.

- This is internal fragmentation.
- Placement strategies have been devised to help minimize the unused space in a file.

PLACEMENT STRATEGIES

- Look for the empty space that exactly fits. Put the record at the end if none is found.
- Look for the empty space that is larger or equal. If larger make a small piece of what is left over.
- In either of these we can keep the empty space list sorted in either ascending or descending order to help find a spot that the new record fits in best.
- Neither of these really works well.
- You can also choose the first empty space you find
- Then use some other technique to deal with the left over space
    - Compaction: should get this in OS
    - Combine adjacent unused space
    - Periodically rebuild the file

COMBINE ADJACENT SPACE

Used Unused

0	32	08Zaphod	10Trillium	06Ford	08Marvin	
		0	8	18	24	32

Delete Trillium

Used Unused

0	8	08Zaphod	10*32 8	06Ford	08Marvin	
		0	8	18	24	32

Delete Ford

Used Unused

0	8	08Zaphod	16*32 8	08Marvin	
		0	8	24	32

Add Arthur

Used Unused

0	16	08Zaphod	08Arthur	08*32 16	08Marvin	
		0	8	16	24	32

- Now the space left is larger
- At the delete you can look both ways but this requires more record keeping.
- You still get internal fragmentation especially if you don't look both ways.

Compaction and Pointers

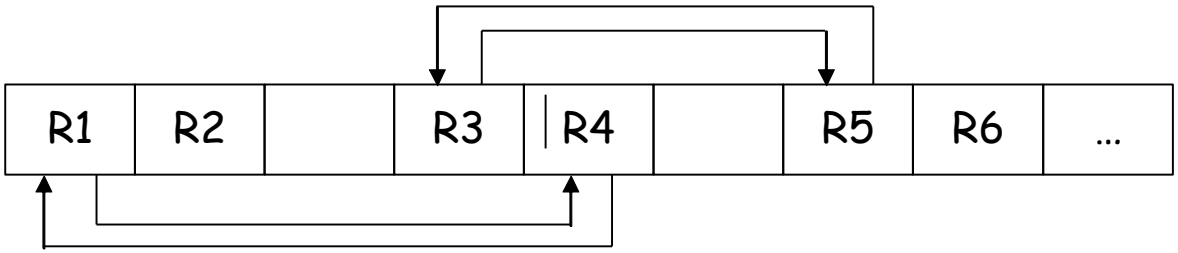
- If you use compaction you must be careful if there are pointers in the file.
- You have to make sure they are changed to reflect the movement of the records.

R1	R2		R3	R4		R5	R6	…

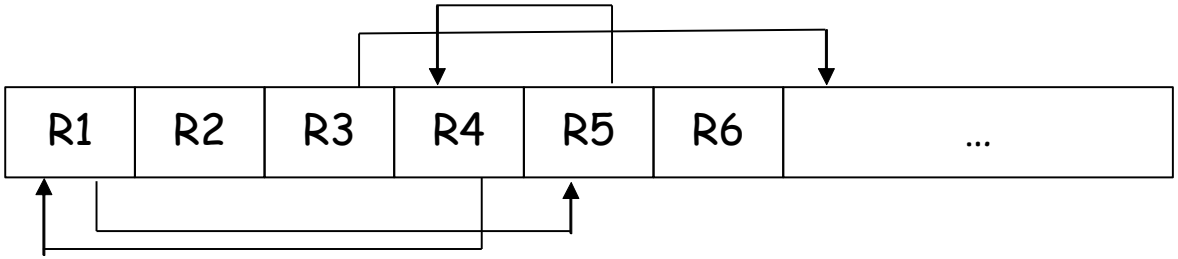
Compaction without pointers is easy. Just move the records.

R1	R2	R3	R4	R5	R6	…

IF the pointers are absolute addresses we have the following.
What happens if they are relative addresses?



Compaction with pointers requires changing the pointers as well.
So, we don't use pointers we use indices and keys.



SORTING and SEARCHING

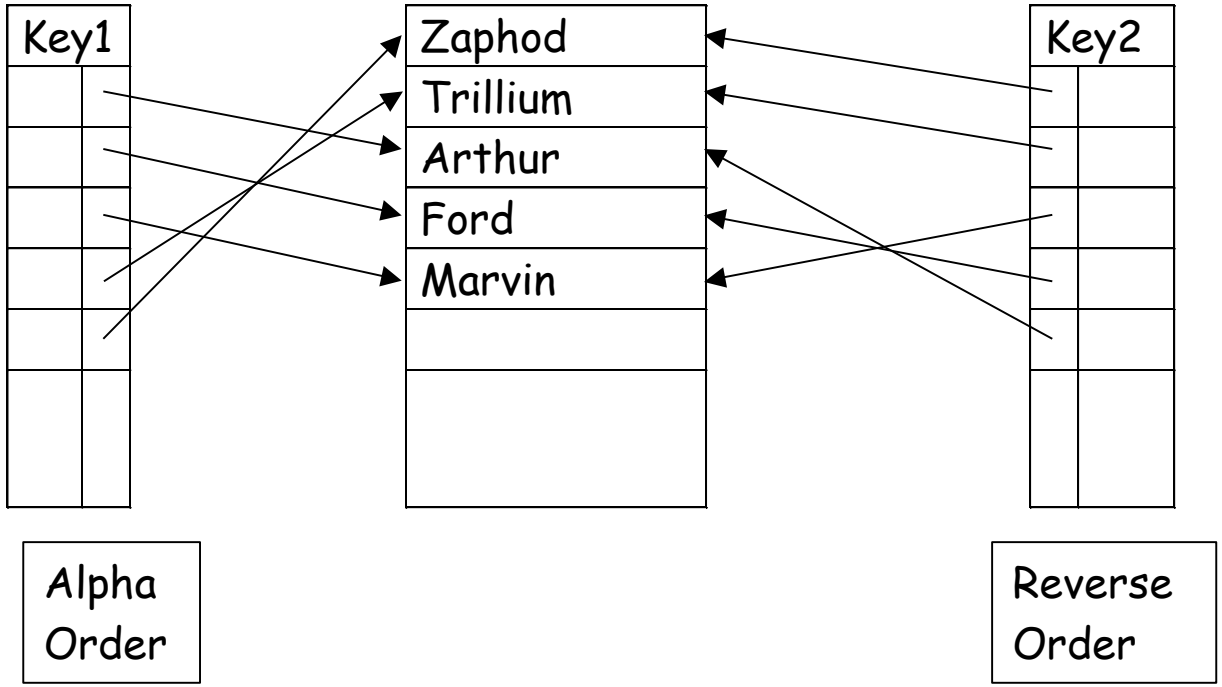
If the list is sorted, you can use a binary search to find a given record.

# of Records	Sequential		Binary
	Average	Worst	Worst
5000	2500	5000	13
10000	5000	10000	14
20000	10000	20000	15
50000	25000	50000	16
100000	50000	100000	17

This is great but how can you sort the list?

The file must fit in memory.

- We are going to talk about how to solve this problem.
- But first we need to talk about KEYS and INDICES
- We can leave the records in place and sort the INDICES



- This way you can use as many keys as you like and consider the data in any way you like. Alpha Order, SSN Order, TechID Order