Overview: File and Indexing

File Storage:

1. Physical storage hierarchy
2. Secondary storage
3. Storage access
4. Redundant arrays of independent disks (RAID)
5. Organize records in a file

Reading materials:

- Chapter 12, Database System Concepts, 7th Edition
- Chapter 16-17, Fundamentals of Database Systems

Organize Records in a File

An operating system (OS) manages a *file system*, where a file is stored on a number of disk blocks.

- A database is stored as a collection of files.
- A file contains a sequence of records.
- A record contains a sequence of fields.

student ID	student name	gpa	...
1	Tom	3.7	...
2	David	3.6	...
3	Jack	3.9	...
4	Eva	4.1	...
5	Tony	2.7	...
6	Adam	3.8	...
...	...	...	...

Organize Records in a File

- Blocking factor: the average number of records per block in a file.

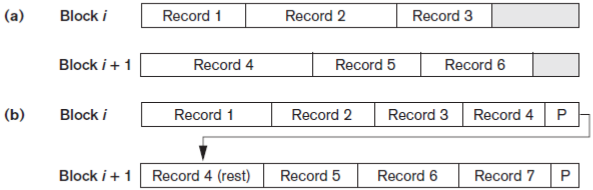


Figure: Types of record organization (a) Unspanned (b) Spanned

We use unspanned organization by default in the future discussions.

Example: Blocking Factor

Blocking factor: the average number (bfr) of records per block in a file.

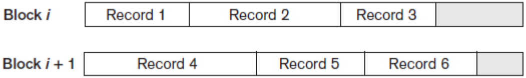


Figure: Unspanned Record Storage

Suppose that we have

- A file with r = 300 records,
- Block size $B = 4096$ bytes,
- Each record is of size $R = 100$ bytes.

- $bfr = \lfloor \frac{B}{R} \rfloor = \lfloor \frac{4096}{100} \rfloor = 40.$
- $\#$ of blocks $= \lceil \frac{r}{bfr} \rceil = \lceil \frac{300}{40} \rceil = 8.$

Organize Records in a File

Given a field X, records can be organized, in a file:

- Ordered (sequential) on field X (X is called the ordering field)
- Unordered (heap) file

Possible operations on a file containing records:

- Search
- Insertion
- Deletion
- Update

Organize records in a file

- Records and blocking
- Sequential files
- Unordered files

Sequential File

Sequential file of EMPLOYEE with the *ordering field* Name:

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
			⋮			
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
			⋮			
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
			⋮			
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
			⋮			
	Anderson, Rob					
Block 5	Anderson, Zach					
	Angeli, Joe					
			⋮			
	Archer, Sue					
Block 6	Arnold, Mack					
	Arnold, Steven					
			⋮			
	Atkins, Timothy					
			⋮			
Block *n*−1	Wong, James					
	Wood, Donald					
			⋮			
	Woods, Manny					
Block *n*	Wright, Pam					
	Wyatt, Charles					
			⋮			
	Zimmer, Byron					

Search

Search for a record with $X = K$ in a file with b blocks ordered on X.

- Sequential scan.
- Binary search (the location of the i-th record can be calculated).

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
Block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
Block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
⋮						
Block $n-1$	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
Block n	Wright, Pam					
	Wyatt, Charles					

Sequential File

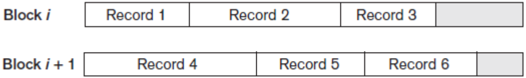
Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

Figure: Average access times for a sequential file of b blocks

Suppose that we have

- An ordered file with $r = 300$ records, the block size $B = 4096$ bytes, records are of fixed size $R = 100$ bytes.
- $bfr = 40$, the file is stored on $b = \lceil \frac{r}{bfr} \rceil = 8$ blocks.
- Worst-case # of I/Os in a binary search? $\lceil \log_2 b \rceil = 3$ I/Os.

Sequential File



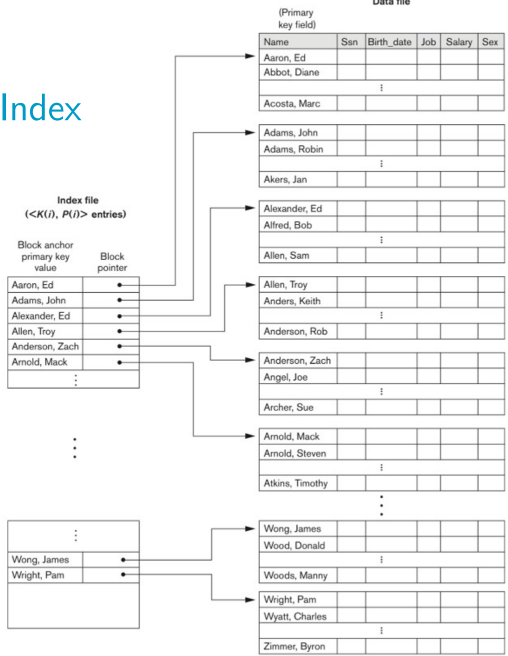
- Search: can we do better than binary search?
- How to handle insertion/deletion?

Index on Sequential Files

- Index Entry ⟨key, block pointer⟩: represent the blocks in a file using a set of ⟨key, block pointer⟩ pairs — each pair is called an *index entry*.

If the *ordering field* is key, the index is called *primary index*.

Primary Index



In a block, the record whose key goes to the index entry is called the *anchor record* or *block anchor*.

Primary Index - Example

Suppose that we have

- an ordered file with $r = 300000$ records,
- the block size $B = 4096 bytes$,
- records are of fixed size $R = 100$ bytes,
- so the blocking factor is $bfr = 40$,
- so the file has $n_b = \lceil \frac{r}{bfr} \rceil = 7500$ blocks,
- a primary index on key X.

Answer the following questions:

- How many index entries (<key, block pointer> pairs) does the index file have?
- If each index entry takes $R_i = 4$ bytes, how many entries can each block hold?
- How many blocks does the index file have?

Primary Index - Example

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

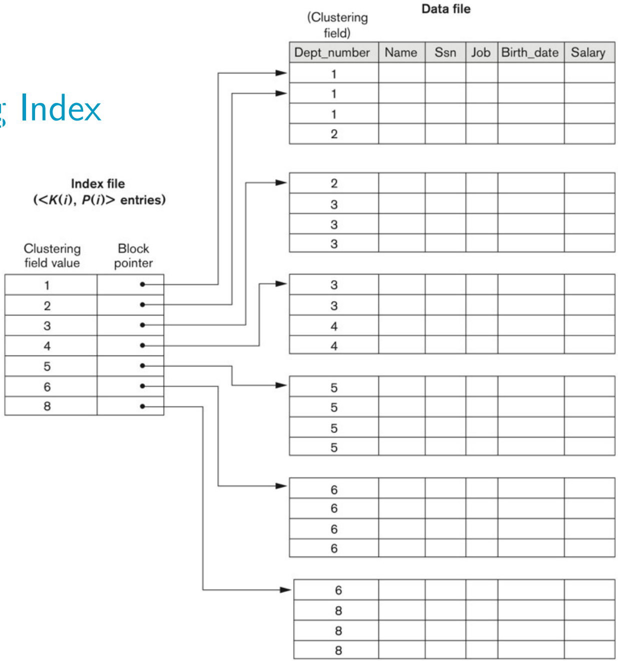
- How many index entries n_i (<key, block pointer> pairs) does the index file have?
  $n_i = n_b$, one entry per data block.
- If each index entry takes $R_i = 4$ bytes, how many entries can each block hold?
  $bfr_{idx} = \lfloor \frac{B}{R_i} \rfloor = 1024$.
- How many blocks does the index file have? $b_{idx} = \lceil \frac{n_i}{bfr_{idx}} \rceil = \lceil \frac{7500}{1024} \rceil = 8$.

Index on Sequential Files

- Index Entry ⟨key, block pointer⟩: represent the blocks in a file using a set of ⟨key, block pointer⟩ pairs — each pair is called an *index entry*.

If the *ordering field* is a key, we build *primary index*.
If the *ordering field* is not a key, we build *clustering index*.

Clustering Index



Organize records in a file

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

- Records and blocking
- Sequential file
- Unordered file

Unordered file

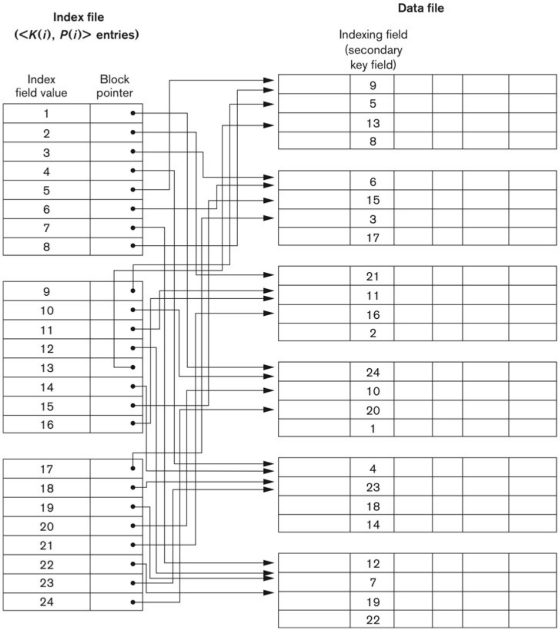
```
Select studentID From Student
Where studentName = `Jack' and `GPA < 3'
```

student ID	student name	gpa	···
1	Tom	3.7	···
2	David	3.6	···
3	Jack	3.9	···
4	Eva	4.1	···
5	Tony	2.7	···
6	Adam	3.8	···
···	···	···	···

- A file that is unordered w.r.t. the indexing field may be ordered by other fields.
- If the indexing field is a key, we build a *secondary index* with *only* index entries.

Unordered File

Secondary index on key



Example: Unordered File Index on Key Values

Suppose that we have

- an unordered file with $r = 300000$ records,
- the block size $B = 4096 bytes$,
- a secondary index on the unordered field where the field is a key,
- each index entry: $s_i = 4$ bytes.
- the number of distinct values in the indexing field: $n_{dis} = 102400$.

Answer the following questions:

- How many blocks does the index file occupy?
  $b_{idx} = n_{dis}/(B/s_i) = 102400/(4096/4) = 100$
- How to retrieve a record of key K? $O(\log_2 b_{idx})$.

Single Level Ordered Indexes

Table 17.2 Properties of Index Types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

[a] Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b] For option 1.
[c] For options 2 and 3.

- Dense index: one index entry for *each* record in the data file
- Non-dense (spase) index: one index entry for a set of records in the data file

Organize records in a file

- Records and blocking
- Sequential file
- Unordered file

Overview: File and Indexing

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

- Tree-based indexes
  - Multi-level indexes
  - Dynamic multi-level indexes — B^+ - tree
- Hashing
  - Static hashing
  - Dynamic hashing — extendible hashing

Reading materials:

- Chapter 13, Database System Concepts, 7th Edition
- Chapter 16-17, Fundamentals of Database Systems

`http://codex.cs.yale.edu/avi/db-book/`

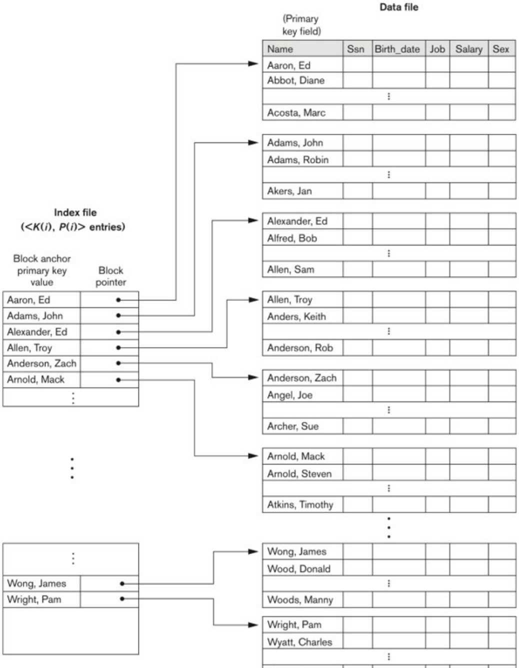
Revision

Single-Leveled Indexes:

- Primary index (key, ordered)
- Clustering index (non-key ordered)
- Secondary index (unordered)

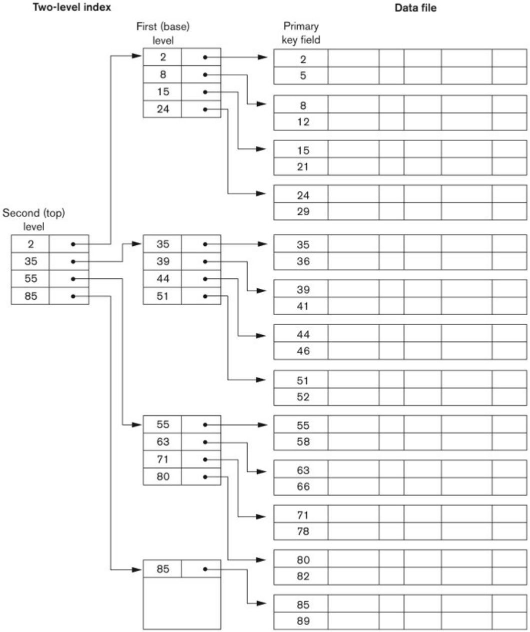
Search: binary search.

Revision



Data file
(Primary key field)
Name
Ssn
Birth_date
Job
Salary
Sex
Aaron, Ed
Abbot, Diane
Acosta, Marc
Adams, John
Adams, Robin
Akers, Jan
Alexander, Ed
Alfred, Bob
Allen, Sam
Allen, Troy
Anders, Keith
Anderson, Rob
Anderson, Zach
Angel, Joe
Archer, Sue
Arnold, Mack
Arnold, Steven
Atkins, Timothy
Wong, James
Wood, Donald
Woods, Manny
Wright, Pam
Wyatt, Charles
Index file
(<K(i), P(i)> entries)
Block anchor primary key value
Block pointer
Aaron, Ed
Adams, John
Alexander, Ed
Allen, Troy
Anderson, Zach
Arnold, Mack
Wong, James
Wright, Pam

Multi-Level Index



Multi-Level Indexes

- Fanout: the blocking factor of the index file, denoted as f.
- If the first level has r_1 index entries,
- The second level has $r_2 = \lceil r_1/f \rceil$ index entries,
- The third level has $r_3 = \lceil r_2/f \rceil$ index entries,
- $\ldots$
- The top index level has one block of at most f index entries.
- There will be approximately $t = \lceil \log_f(r_1) \rceil$ levels.

Example

Consider a multi-level index of a data file. Assume that the fanout $f_i = 273$, the first level index has $b_1 = 1099$ blocks. The secondary level blocks $b_2 = \lceil b_1/f \rceil = 5$. The third level blocks $b_3 = \lceil b_2/f \rceil = 1$. How many blocks should one access at most in order to locate a record with a given key value?

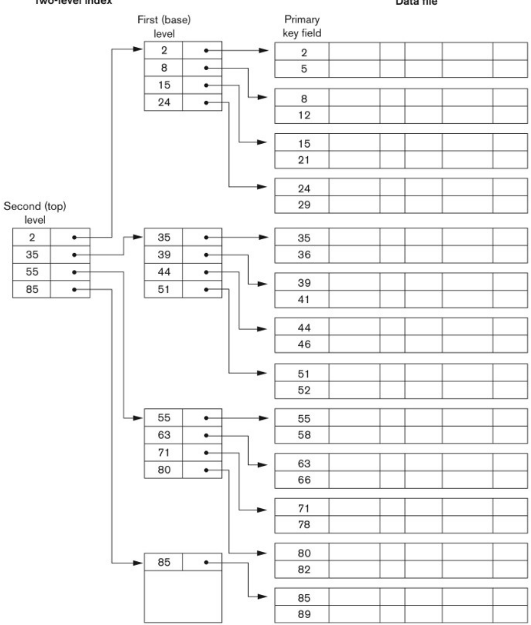
$O(\log_f b_1) = O(\log_{273} 1099)$.

When $b_1 = 1099^2$, the cost becomes $O(2\log_{273} 1099)$.

The query complexity grows extremely slow with the increasing data size.

Search on Multi-Level Indexes

Search: 46



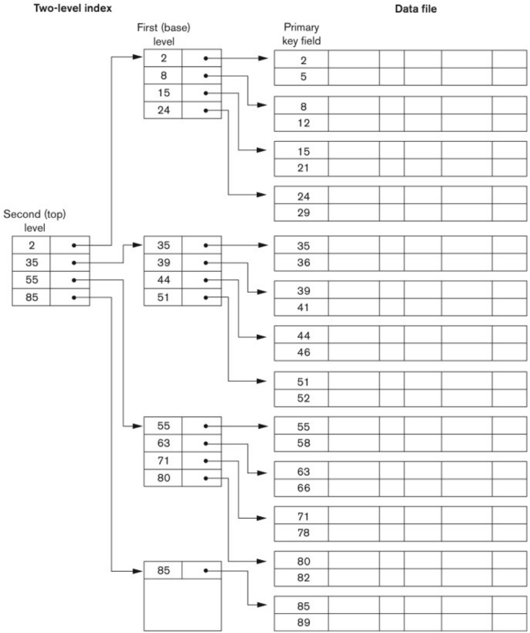
Multi-Level Indexes

How to handle

- insertions?
- deletions?

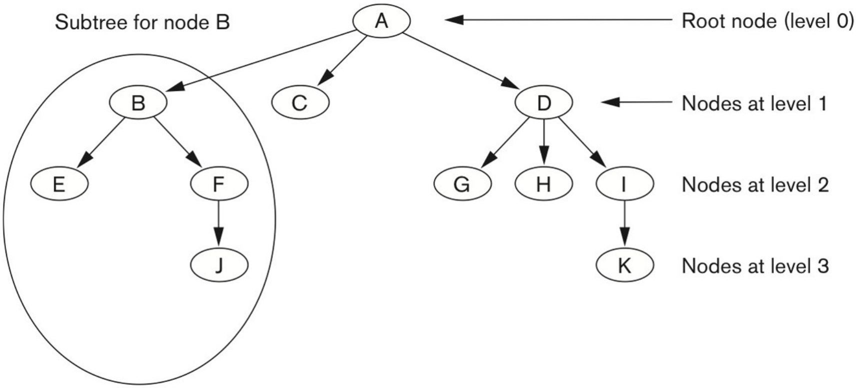
Introduce the concept of

- tree,
- search tree,
- and balanced search tree.



Tree

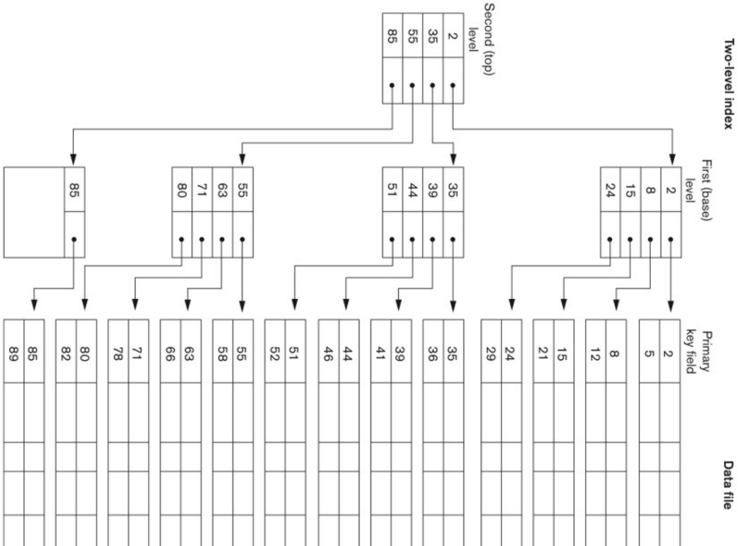
- Nodes
- Parent/Child
- Root/Leaf
- Internal node
- Level
- Subtree
- Balanced tree:
  All leaf nodes are of the “same” level



(Nodes E, J, C, G, H, and K are leaf nodes of the tree)

Multi-Level Index is a Balanced Tree

- Nodes
- Parent/Child
- Root/Leaf
- Internal node
- Level
- Subtree
- Balanced tree: All leaf nodes are at the "same" level

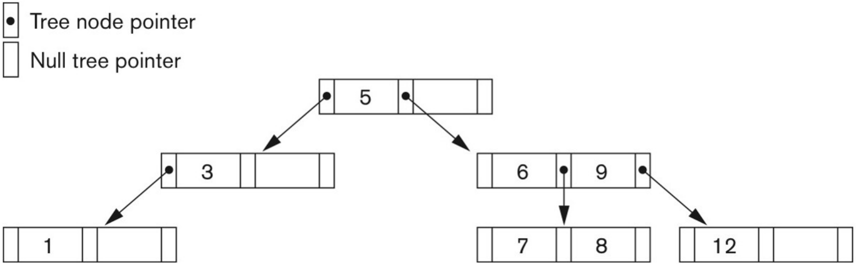


Search Tree

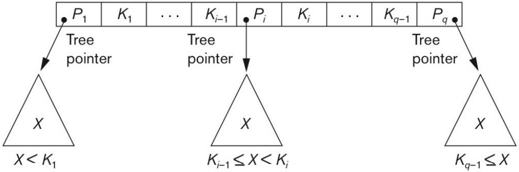
For any node v on a search tree whose value is denoted as K,

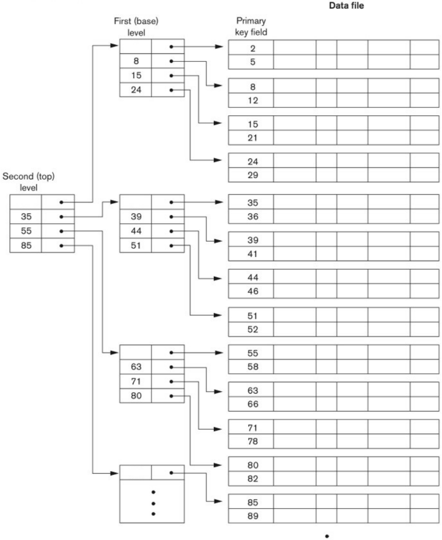
- the values in the left subtree if v is no greater than K;
- the values in the right subtree of v is no smaller than K.

Example of a Search Tree:



Multi-Level Index: a Balanced Search Tree





- Search 7, ranges $[30, 50]$, $[30, +\infty)$
- Insert
- Delete

B^+-Tree: a Dynamic Multi-Level Index

Generally, B^+-Tree requires that all leaves are of the same level, each node (except for the root node) is at least *half full*. An insertion on a full node incurs a split while a deletion from a half-full node causes a redistribution/merge over the current node and its neighbors. The split and merge operation can be triggered recursively.

B^+-Tree is I/O aware, and support search exact and range queries:

- Make 1 node $=$ 1 physical page
- Balanced, height adjusted tree
- Make leaves into a linked list (for range queries)

```
SELECT cname FROM   Company WHERE  price = 25;
SELECT cname FROM   Company WHERE  20 <= price   AND  price <= 30
```

B^+-Tree

B^+-tree with two parameters

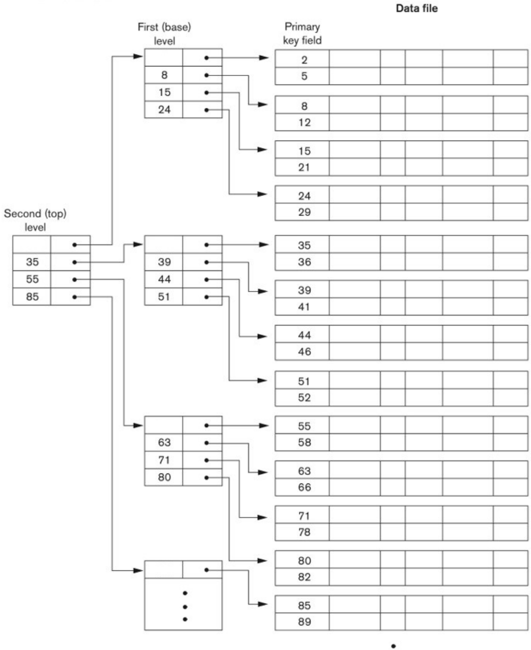
- Blocking factor of the index files (index fanout) f_i and
- Blocking factor of the data files f_l.

Each leaf node contains

- at most f_l data records
- at least $\lceil f_l/2 \rceil$ data records

Each internal node (except the root) contains

- f **tree pointers** and $f - 1$ keys (not records!!!) where
- f is at most f_i and
- f is at least $\lceil f_i/2 \rceil$



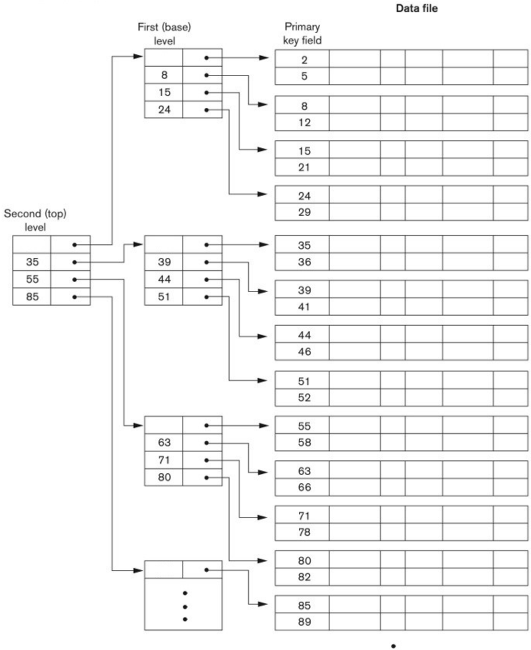
B^+-Tree

B^+-tree with two parameters

- Blocking factor of the index files (index fanout) f_i and
- Blocking factor of the data files f_l.

Facts that you should know:

- How many search values per page?
- How many levels in tree?
- What is the I/O cost of an equal search?
- What is the I/O cost of a range search?
- What is the I/O cost of an update?



B^+-Tree: Cost Analysis

Let:

- f =fanout, which is in $[f_i/2, f_i]$
- N = the total number of pages we need to index
- F = fill-factor (usually $= 2/3$)

What height (h) does our B^+ Tree need to be?

$h = 2$: Just the root node - room to index f pages

$h = 3$: f level-1 nodes - room to index f^2 pages

$h = 4$: f^2 level-1 nodes - room to index f^3 pages

...

h : f^{h-2} level-1 nodes - room to index f^{h-1} pages!

$h = O(\log_f N)$.

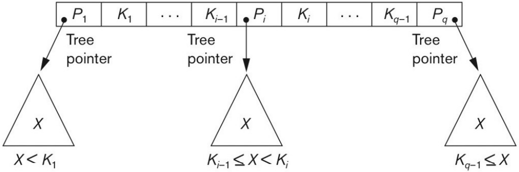
B^+-Tree: Cost Analysis

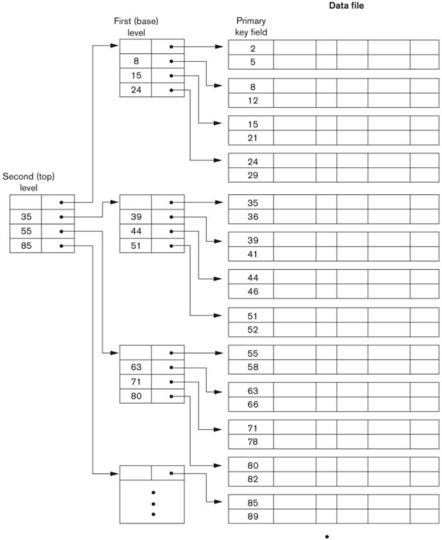
Example:

- Key size $= 4$ bytes, Pointer size $= 8$ bytes
  Note: record size not relevant for these calcs (could be MBs)
- We want each node to fit in a single block/page (4 KBs)
  $2f \times 4 + (2f + 1) \times 8 \leq 4k => f = 170$
- What is the height of a B^+ Tree that indexes 10^8 pages? $h = 5$.

B^+-Tree: Insertion

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND
SCIENCE
DEPARTMENT OF COMPUTER SCIENCE





- Search 7, ranges $[30, 50]$, $[30, +\infty)$
- Insert
- Delete

B^+-Tree: Insertion

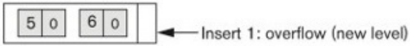
1. Search K, then
2. Insert K

Consequences:

- A leaf node is overflow if it has more than f_l data records
- An internal node is overflow if it has more than f_i pointers

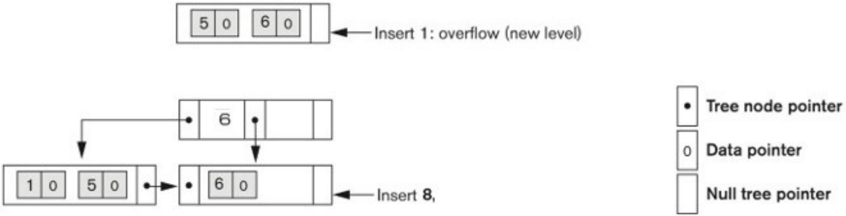
Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1



Example: $f_i = 3, f_l = 2$

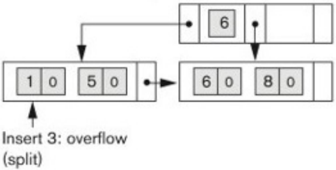
Insert 6, 5, 1, 8

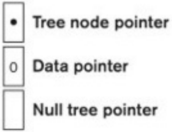


Evenly split; the size of first node splitted is no smaller than the second node splitted!
A leaf registers its first key to the index entry in the parent.

Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1, 8, 3

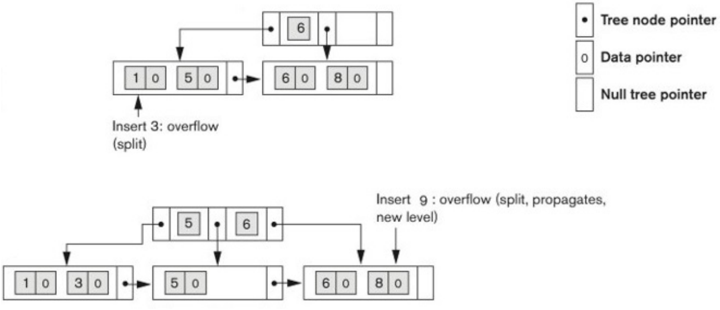




Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1, 8, 3:

- insertion (overflow) $\boxed{1,5}$ => $\boxed{1,3,5}$
- evenly split $b = \boxed{1,3,5}$ => $\boxed{1,3}$ and $\boxed{5}$
- register $\boxed{1,3}$, 5, $\boxed{5}$ to the parent of b, 5 is the key of the anchor record of node $\boxed{5}$

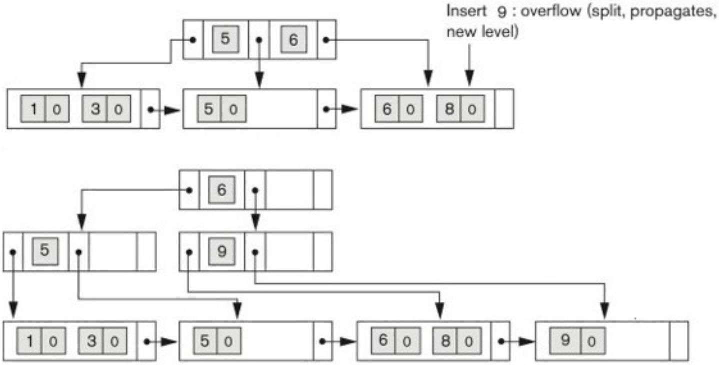


Example: $f_i = 3, f_l = 2$ - Recursive Split

Insert 6, 5, 1, 8, 3, 9

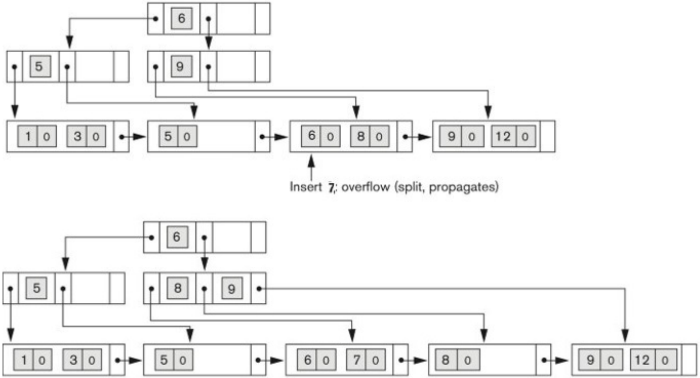
Steps:

1. insert 9 to leaf $\boxed{6,8}$
2. partition the overflowed leaf $\boxed{6,8,9} \Rightarrow \boxed{6,8}$ & $\boxed{9}$
3. insert $\boxed{6,8}$ 9 $\boxed{9}$ to the parent $\boxed{5,6}$
4. partition the overflowed root $\boxed{5,6,9} \Rightarrow \boxed{5}$ 6 $\boxed{9}$
5. put 6 as the single key in the new root with left child $\boxed{5}$ and right child $\boxed{9}$



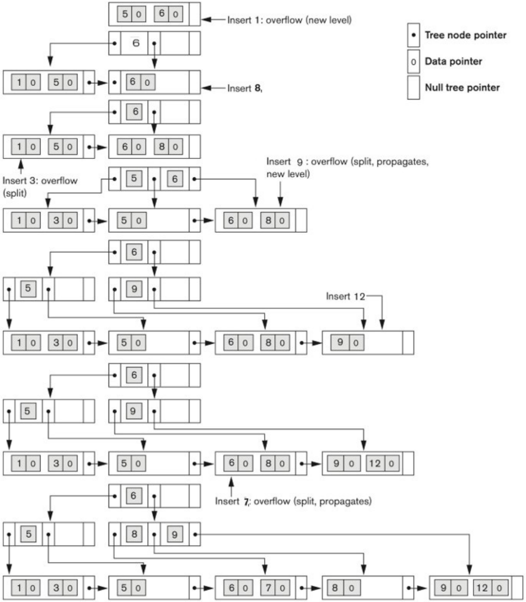
Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1, 8, 3, 9, 12, 7



Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1, 8, 3, 9, 12, 7



B^+-Tree: Insertion

Insert a data record with key K:

1. p: find the address of the data block of K;
2. b: load p into the main memory;
3. insert the record to b;
4. if b overflows,
   - split b into two new blocks with addresses c_1 and c_2:
     - c_1 contains the first (smallest) $\lceil f_l + 1 \rceil$ records
     - c_2 contains the last (largest) $\lfloor f_l + 1 \rfloor$ records
   - let s be the anchor record of c_2;
   - if p was the root before the insertion, let the new root be $\boxed{c_1, s, c_2}$; otherwise,
     - let *anc* be the parent of p,
     - call register(anc, p, s, c_1, c_2) — a procedure that registers the split of p to its parent
5. otherwise, write b back to address p.

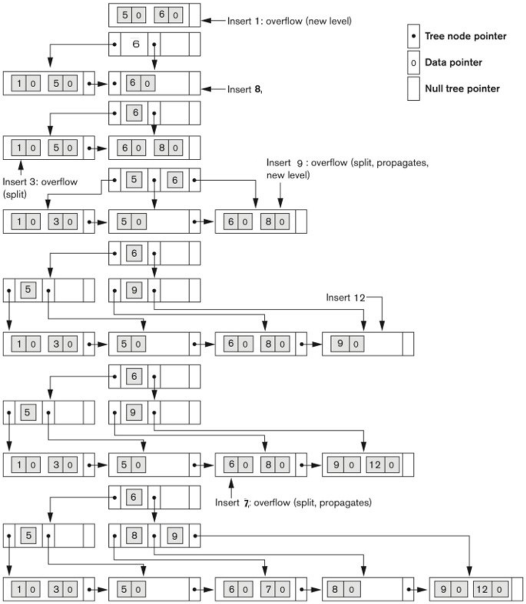
B^+-Tree: Insertion

register(anc, p, s, p_1, p_2):

1. Let $K_1, P_1, K_2, P_2, \cdots, K_{n-1}, P_n$ be the content of the block of anc;
2. Let i be the integer such that $P_i = p$;
3. Register the split by letting $b = \boxed{K'_1, P'_1, K'_2, P'_2, \cdots, K'_n, P'_{n+1}}$ be $\boxed{K_1, P_1, \cdots, K_i, p_1, s, p_2, K_{i+1}, \cdots, K_{n-1}, P_n}$.
4. if b overflows,
   - split b into two new blocks with addresses c_1 and c_2:
     - let $m = \lfloor (n+1)/2 \rfloor$
     - c_1 contains $\boxed{P'_1, K'_1, \cdots, K'_{m-1}, P'_m}$; c_2 contains $\boxed{P'_{m+1}, K'_{m+2} \cdots K'_n, P'_{n+1}}$
   - let s be the anchor record of K_{m+1};
   - if anc is the root, then let the new root be c_1, s, c_2 otherwise,
     - let anc' be the parent of anc,
     - call register(anc', anc, s, c_1, c_2)
5. otherwise, write b back to address anc.

Example: $f_i = 3, f_l = 2$

Insert 6, 5, 1, 8, 3, 9, 12, 7
Delete 7, 12, 9, 3, 8, 1, 5, 6
Follow the code



Overview: Indexes

- Tree-based indexes
  - Multi-level indexes
  - Dynamic multi-level indexes — B^+ - tree
- Hashing
  - Static hashing
  - Dynamic hashing — extendible hashing

Reading materials:

- Chapter 11, Database System Concepts, 7th Edition
- Chapter 16-17, Fundamentals of Database Systems

http://codex.cs.yale.edu/avi/db-book/