Indexing

Introduction to Databases

CompSci 316 Spring 2020

DUKE
COMPUTER SCIENCE

1

Announcements (Thu., Feb 27)

- Private project threads created on piazza
  - Please check you are on your thread
  - Primary and secondary project mentors to be assigned soon
  - They will give you feedback on MS1, check updates, and help you as needed
  - Feel free to discuss projects in all TA office hours
- Project updates to be posted **every Monday**
  - Starts Monday 03/02: each member should say what you are supposed to do for the rest of the semester and also for MS2
  - Make sure that your primary TA says "sounds good" in the response to each update, otherwise do as they suggest
  - Other team members will also check the updates and respond to the threads as needed if there are confusions or clarifications
  - Try to resolve conflicts/concerns within group whenever possible, otherwise reach out to your TAs and me early

2

Announcements - contd (Thu., Feb 27)

- Homework #4 published due next Wednesday 03/04
  - One submission per project group
  - See updates on piazza about submitting the link to your website
- Let me know if you want to meet me about midterm or anything else
  - Final exam will be similar in nature (problem-solving based), but the length/difficulty/question types may vary
  - Comprehensive with more focus on topics after midterm
  - How to prepare? Think about what we discuss in class and ask me tough questions!
- Heads up: (almost) weekly quiz or lab **every Thursday**
  - For practicing problems for the final
  - In groups, but individual submissions
  - Quizzes are shorter and discussed in class, Labs are longer with extra time after class (extra credit for submitting within the class)

3

Today's lecture

- Index
- Dense vs. Sparse
- Clustered vs. unclustered
- Primary vs. secondary
- Tree-based vs. Hash-index

(Dense vs. Sparse, Clustered vs. unclustered, Primary vs. secondary: Related)

4

What are indexes for?

- Given a value, locate the record(s) with this value

  SELECT * FROM *R* WHERE *A = value*;

  SELECT * FROM *R, S* WHERE *R.A = S.B*;
- Find data by other search criteria, e.g.
  - Range search

    SELECT * FROM *R* WHERE *A > value*;
  - Keyword search

(Focus of this lecture)

[database indexing] [Search]

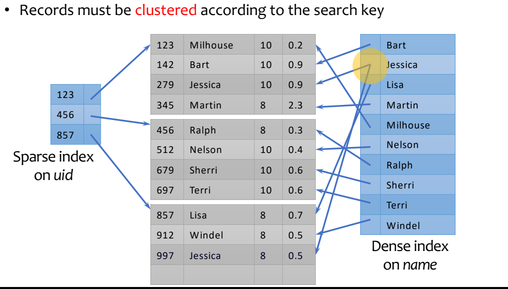
5

Dense and sparse indexes

When are these possible?

Comparison?

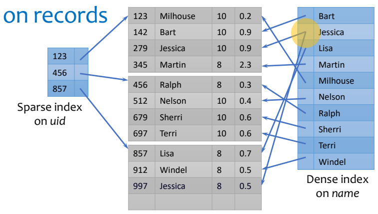
- Dense: one index entry for each search key value
  - One entry may "point" to multiple records (e.g., two users named Jessica)
- Sparse: one index entry for each block
  - Records must be clustered according to the search key



6

Dense versus sparse indexes

- Index size
  - ??
- Requirement on records
  - ??
- Lookup
  - ??
- Update
  - ??



7

Dense versus sparse indexes

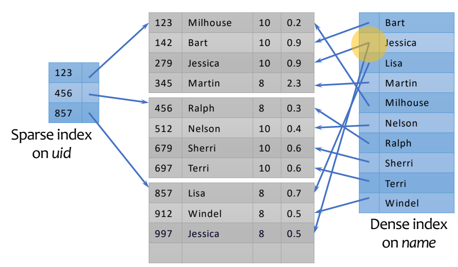
8

Primary and secondary indexes

- Primary index
  - Created for the primary key of a table
  - Records are usually clustered by the primary key
  - Can be sparse
- Secondary index
  - Usually dense
- SQL
  - PRIMARY KEY declaration automatically creates a primary index, UNIQUE key automatically creates a secondary index
  - Additional secondary index can be created on non-key attribute(s):
    CREATE INDEX UserPopIndex ON User(pop);

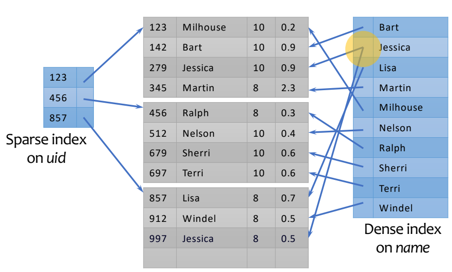
9

What if the index is too big as well?



10

What if the index is too big as well?

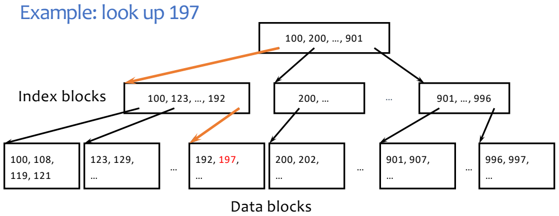


11

ISAM

☞ISAM (Index Sequential Access Method), more or less

Example: look up 197

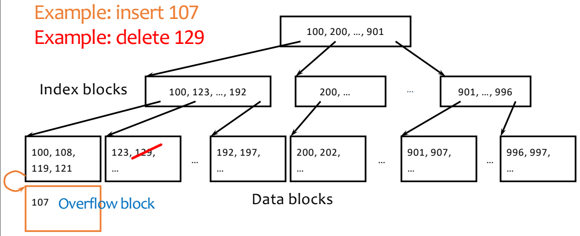


12

Updates with ISAM

13

Example: insert 107
Example: delete 129



- Overflow chains and empty data blocks degrade performance
  - Worst case: most records go into one long chain, so lookups require scanning all data!

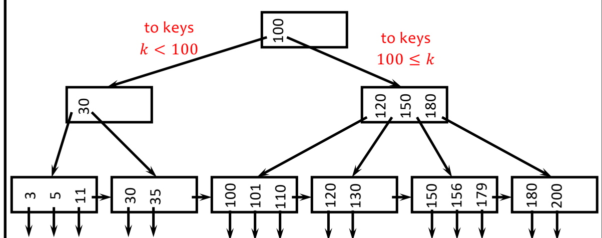
13

B+-tree

14

Max fan-out: 4

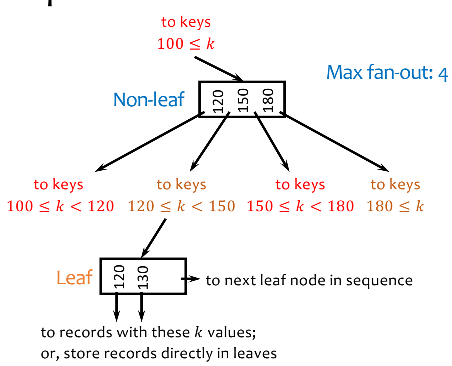
- A hierarchy of nodes with intervals
- Balanced (more or less): good performance guarantee
- Disk-based: one node per block; large fan-out



14

Sample B+-tree nodes

15



15

B+-tree balancing properties

16

Check yourself

- Height constraint: all leaves at the same lowest level
- Fan-out constraint: all nodes at least half full (except root)

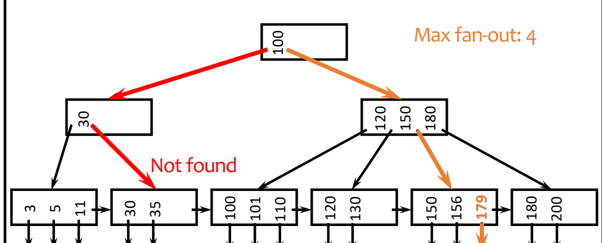
	Max # pointers	Max # keys	Min # active pointers	Min # keys
Non-leaf	f	$f-1$	$\lceil f/2 \rceil$	$\lceil f/2 \rceil - 1$
Root	f	$f-1$	2	1
Leaf	f	$f-1$	$\lfloor f/2 \rfloor$	$\lfloor f/2 \rfloor$

16

Lookups

17

- SELECT * FROM *R* WHERE *k* = 179;
- SELECT * FROM *R* WHERE *k* = 32;

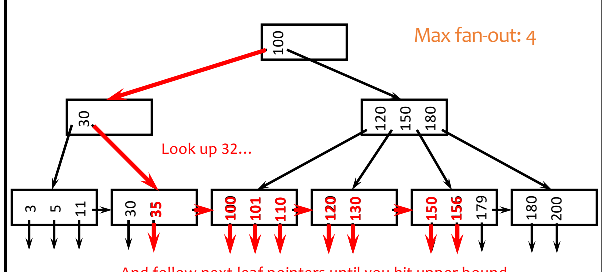


17

Range query

18

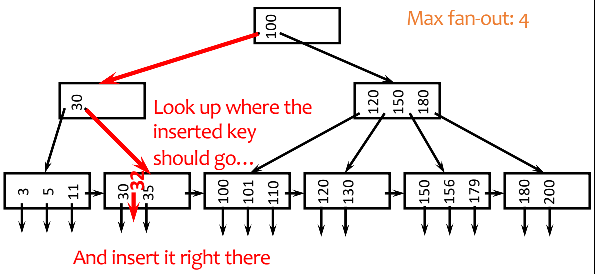
- SELECT * FROM *R* WHERE *k* > 32 AND *k* < 179;



18

Insertion

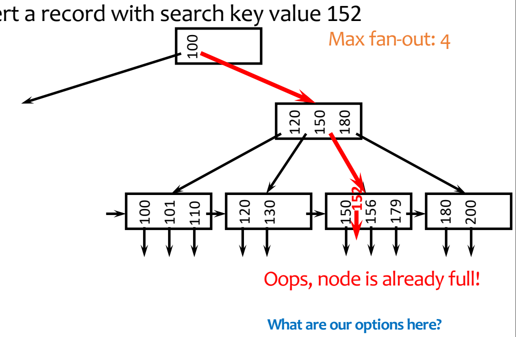
- Insert a record with search key value 32



19

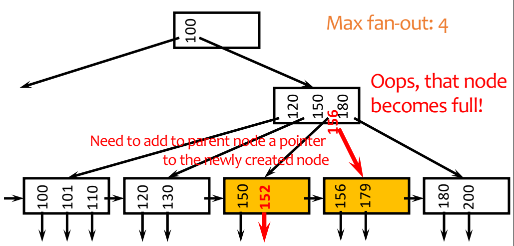
Another insertion example

- Insert a record with search key value 152



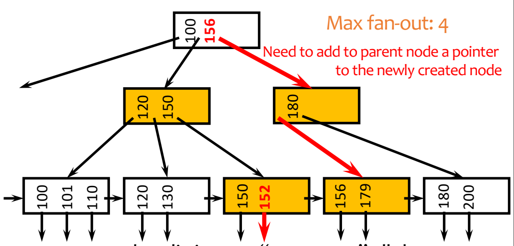
20

Node splitting



21

More node splitting

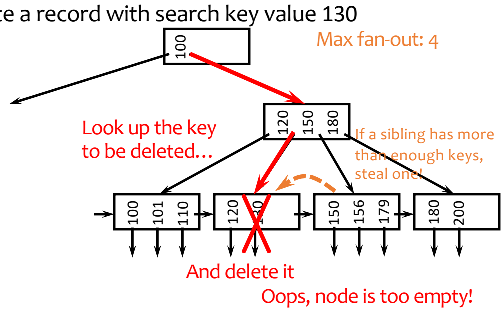


- In the worst case, node splitting can "propagate" all the way up to the root of the tree (not illustrated here)
  - Splitting the root introduces a new root of fan-out 2 and causes the tree to grow "up" by one level

22

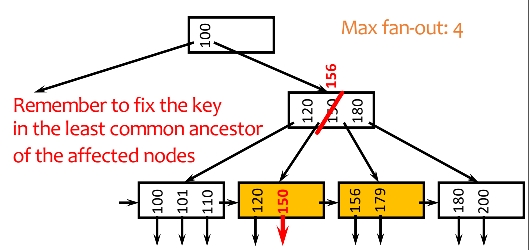
Deletion

- Delete a record with search key value 130



23

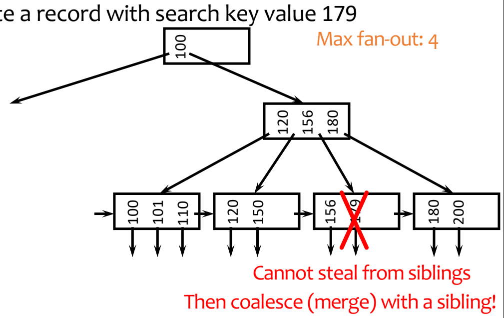
Stealing from a sibling



24

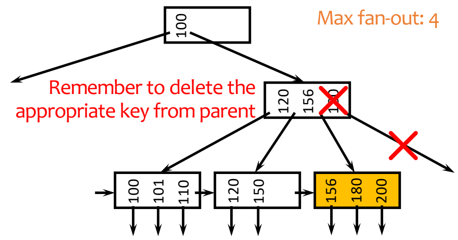
Another deletion example

- Delete a record with search key value 179



25

Coalescing



- Deletion can "propagate" all the way up to the root of the tree (not illustrated here)
  - When the root becomes empty, the tree "shrinks" by one level

26

Performance analysis

- How many I/O's are required for each operation?
  - h, the height of the tree (more or less)
  - Plus one or two to manipulate actual records
  - Plus $O(h)$ for reorganization (rare if f is large)
  - Minus one if we cache the root in memory
- How big is h?

27

B+-tree in practice

- Complex reorganization for deletion often is not implemented (e.g., Oracle)
  - Leave nodes less than half full and periodically reorganize
- Most commercial DBMS use B+-tree instead of hashing-based indexes because B+-tree handles range queries
  - A key difference between hash and tree indexes!

28

The Halloween Problem

- Story from the early days of System R…

```
UPDATE Payroll
SET salary = salary * 1.1
WHERE salary >= 100000;
```

  - There is a B+-tree index on *Payroll*(*salary*)
  - The update never stopped (why?)
- Solutions?

https://en.wikipedia.org/wiki/Halloween_Problem

29

B+-tree versus ISAM

- ISAM is more static; B+-tree is more dynamic
- ISAM can be more compact (at least initially)
  - Fewer levels and I/O's than B+-tree
- Overtime, ISAM may not be balanced
  - Cannot provide guaranteed performance as B+-tree does

30

B+-tree versus B-tree

- B-tree: why not store records (or record pointers) in non-leaf nodes?
  - These records can be accessed with fewer I/O's
- Problems?

31

FYI – not covered in this class

Beyond ISAM, B-, and B+-trees

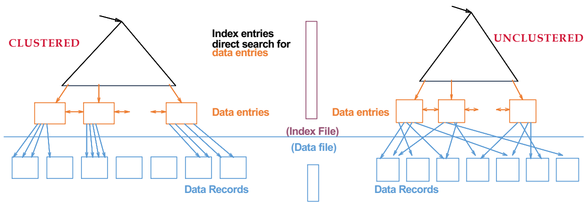
- Other tree-based indexes: R-trees and variants, GiST, etc.
  - How about binary tree?

- Hashing-based indexes: extensible hashing, linear hashing, etc.
- Text indexes: inverted-list index, suffix arrays, etc.
- Other tricks: bitmap index, bit-sliced index, etc.

32

Clustered vs. Unclustered Index

- If order of data records in a file is the same as, or `close to', order of data entries in an index, then clustered, otherwise unclustered
- How does it affect # of page accesses? (in class)



33

Clustered vs. Unclustered Index

- How does it affect # of page accesses? (in class)
- SELECT * FROM USER WHERE age = 50
  - Assume 12 users with age = 50
  - Assume one page can hold 4 User records
  - Suppose accessing the data entry (-ies) require 3 IOs in a B+-tree, which contain pointers to the data records (all pointers in the same node)
  - What happens if the index is unclustered?
  - What happens if the index is clustered?

34

Hash vs. Tree Index

- Hash indexes can only handle equality queries
  - SELECT * FROM R WHERE age = 5 (requires hash index on (age))
  - SELECT * FROM R, S WHERE R.A = S.A (requires hash index on R.A or S.A)
  - SELECT * FROM R WHERE age = 5 and name = 'Bart' (requires hash index on (age, name))
- Cannot handle range queries
  - SELECT * FROM R WHERE age >= 5
  - need to use tree indexes (more common)
  - Tree index on (age), or (age, name) works, but not (name, age) – why?
- + But are more amenable to parallel processing
  - late hash-based join
- Performance depends on how good the hash function is (whether the hash function distributes data uniformly and whether data has skew)
- Details of hash-based dynamic index (extendible hashing, linear hashing) not covered in this class

35

Trade-offs for Indexes

- Should we use as many indexes as possible?

36

Trade-offs for Indexes

- Should we use as many indexes as possible?

37

Index-Only Plans

- A number of queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available

```
SELECT  E.dno, COUNT(*)
FROM  Emp E
GROUP BY  E.dno
```

<E.dno>

```
SELECT  E.dno, MIN(E.sal)
FROM  Emp E
GROUP BY  E.dno
```

<E.dno,E.sal>
Tree index!

<E. age,E.sal>
Tree index!

```
SELECT AVG(E.sal)
FROM  Emp E
WHERE  E.age=25 AND
  E.sal BETWEEN 3000 AND 5000
```

- For index-only strategies, clustering is not important

38