PennState

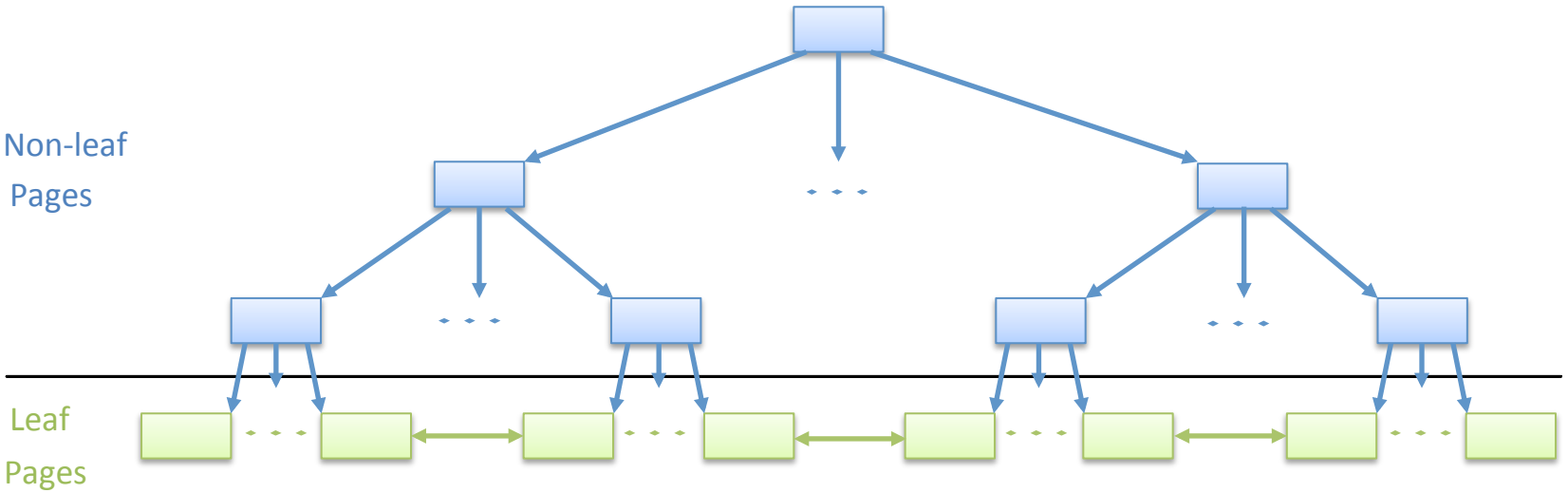
CMPSC431W: Database Management Systems

Lecture 34
11/30/15

Instructor: Yu-San Lin
yusan@psu.edu

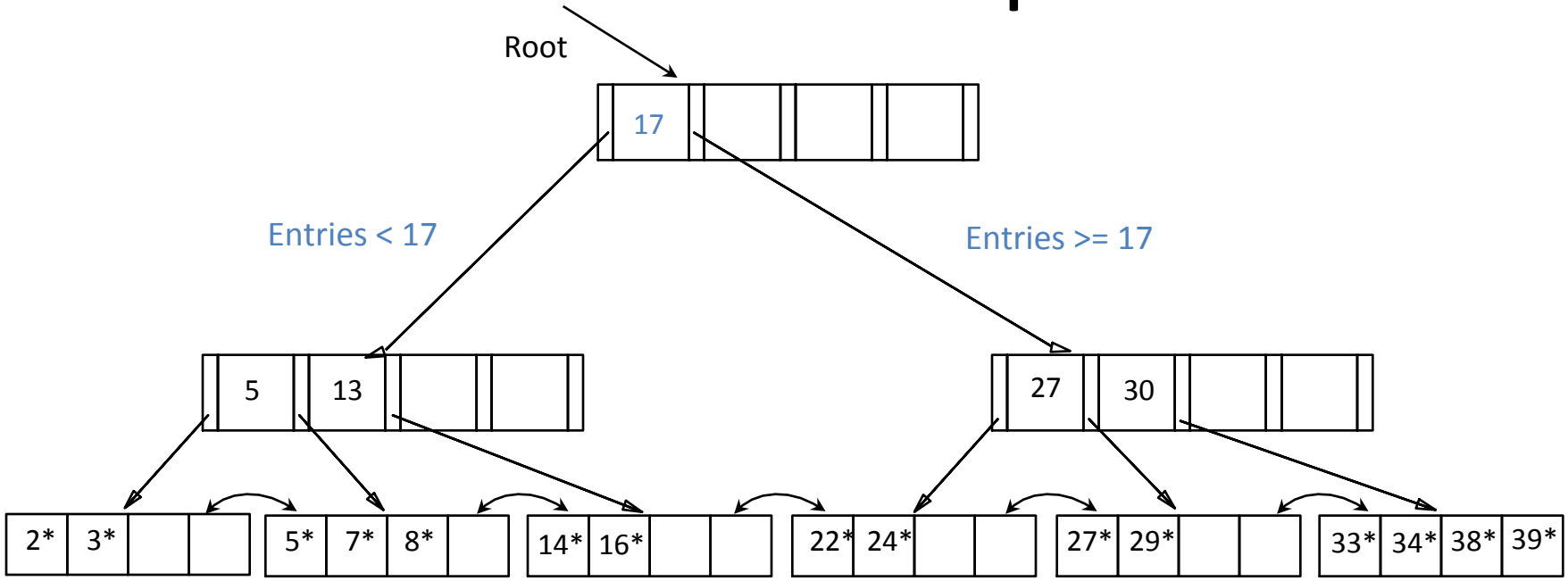
Course Website: http://www.cse.psu.edu/~yul189/cmpsc431w
Slides based on McGraw-Hill & Dr. Wang-Chien Lee

B+ Tree Indexes



- Leaf pages contain data entries, and are chained (prev & next)
- Non-leaf pages contain index entries to direct searches

B+ Tree: Example



- Find:
  - 28*
  - 29*
  - All > 15*
  - All < 30*
- Insert/delete: find data entry in leaf, then change it. Need to adjust parent sometimes
  - And change sometimes bubbles up the tree

Cost Model for Analysis

- For simplicity, ignore CPU costs:
  - B: number of data pages
  - R: number of records per page
  - D: (average) time to read or write disk page
- Average-case analysis; based on several simplistic assumptions

Assumptions in the Analysis

- Heap files
  - Equality selection on key; exactly one match
- Sorted files
  - Files compacted after deletions
- Indexed files
  - Data records (= data entries) are clustered
- Indexes
  - Alternative 2, 3: data entry size = 10% size of record
  - Hash: no overflow buckets
    - 80% page occupancy >= file size = 1.25 data size
  - Tree: 67% occupancy
    - Implies file size = 1.5 data size

Heap Files

- Scan: _____
- Equality select
  - Equality selection on key: _____
  - Search not on a key: _____
- Range select: _____
- Insert
  - Assume always insert at the end of file: _____
- Delete
  - Write the page back: _____
  - If the search is based on a given rid: _____
  - If the search is based on equality or range select: _____

Sorted Files

- Scan: _____
- Equality select: search based on sorted order
  - Match only 1 record: _____
  - Match more than 1 records: _____ _____
- Range select: _____
- Insert: _____
- Delete
  - Read and rewrite all subsequent pages: _____
  - If the search is based on a given rid: ____
  - If the search is based on equality or range selects: _____

Clustered Indexed Files

- Empirical study shows that the pages usually have 67% occupancy. The # of physical pages is: _____
- Scan: _____
- Equality select: search based indexed key
  - Match only 1 record: _____
  - Match more than 1 records: _____ _____
- Range select: _____
- Insert: _____
- Delete: _____

Unclustered Tree Index

- Assume that the size of data entry in the index is 1/10 of data records. If the index pages have 67% occupancy, the # of leaf pages is: ~0.15B. The number of data entries on a page is 6.7R.
- Scan: _____
- Equality select: search based on indexed key
  - Match only 1 record: _____
  - Match more than 1 records: _____ _____

Unclustered Tree Index (cont.)

- Range select: _____
  - If more than 10% of records satisfy the select condition, we are better off retrieve all the data records, sort them based on selection attribute, and retain the records satisfying the selection
- Insert: _____
- Delete: _____

Indexes Selection

- Consider the most important queries in turn
- Consider the best plan using the current indexes, and see if a better plan is possible with an additional index
- Attributes in _____ clause are good index candidates
  - Exact match condition suggests hash index
  - Range query suggests tree index
  - Clustering is especially useful for range queries

Indexes Selection (cont.)

- Multi-attribute search keys should be considered when a WHERE clause contains several conditions
  - Order of attribute is important for range queries
  - Such indexes can sometimes enable index-only query processing (i.e., not accessing data records)

Index Selection: Example #1

```
SELECTC  E.dno
FROM     Emp E
WHERE    E.age > 20 AND E.age < 30
```

- A B+ tree index on E.age can be used to get qualifying tuples
  - How selective is the condition? If everybody is greater than 10, better off to perform a sequential scan
  - Is the index clustered?

Index Selection: Example #2

```
SELECTC E.dno, COUNT(*)
FROM    Emp E
WHERE   E.age > 10
GROUP BY  E.dno
```

- Consider the GROUP BY query
  - If many tuples have E.age > 10, using E.age index and sorting the retrieved tuples by dno is costly
  - Clustered E.dno tree index may be better

Index Selection: Example #3

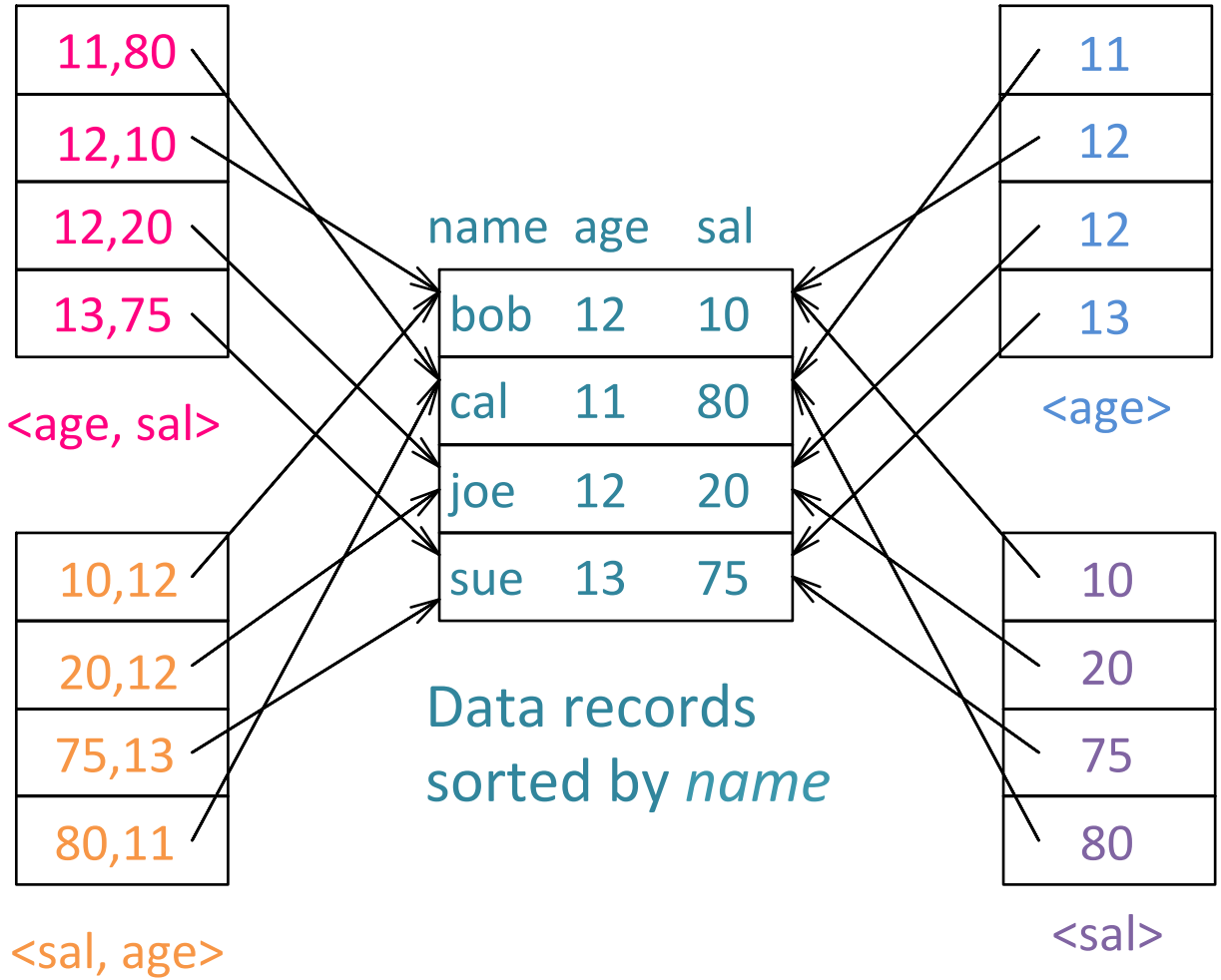
```
SELECTC E.dno
FROM    Emp E
WHERE   E.hobby = Stamps
```

- Equality queries based on non-candidate keys
  - Possibly return duplicates
  - Clustering index on E.hobby helps

Indexes w/ Composite Search Keys

- Composite search keys: search on a combination of fields
  - Equality query: every field value is equal to a constant value. E.g., age = 20 and sal = 75
  - Range query: some field value is not a constant. E.g., age = 20 and sal > 10
- Data entries in index sorted by search key to support range queries
  - Lexicographic order
  - Spatial order

Example of Composite Search Keys



Composite Search Keys

- To retrieve Emp records with age = 30 AND sal = 40
  - Which is a better index?
    (a) <age, sal> (b) age (c) sal
  - Choice of index key orthogonal to clustering etc.
- If condition is: 20 < age < 30 AND 30 < sal <50
  - What is a better index?

- If condition is: age = 30 AND 30 < sal < 50
  - Which is a better index?
    (a) Clustered <age, sal> index (b) clustered <sal, age> index
- Composite indexes are larger updated more often

Index-Only Execution Plans

- Some 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
```

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

```
SELECT     AVG(E.sal)
FROM       Emp E
WHERE      E.age = 25 AND
           E.sal BETWEEN 30 AND 50
```

Index-Only Execution Plans (cont.)

- Index-only plans are possible if we have a tree index with key <dno, age> or with key <age, dno>
- Which is better?

```
SELECT     E.dno, COUNT(*)
FROM       Emp E
WHERE      E.age = 30
GROUP BY   E.dno
```

```
SELECT     E.dno, COUNT(*)
FROM       Emp E
WHERE      E.age >= 30
GROUP BY   E.dno
```

Summary

- Many alternatives file organizations exist, each appropriate in some situation
- If selection queries are frequent, sorting the file or building an index is important
  - Hash-based only good for _____ search
  - Sorted files and tree-based indexes best for range search; also good for equality search
- Index is a collection of data entries plus a way to quickly find entries with given key values

Summary (cont.)

- Data entries can be actual data records, <key, rid> pairs or <key, rid-list> pairs
- Can have several indexes on a given file of data records, each with a different search key
- Indexes can be classified as clustered v.s. unclustered, primary v.s., secondary. Differences have important consequences for utility/performance

Summary (cont.)

- Indexes must be chosen to speed up important queries
  - Indexes maintenance overhead on updates to key fields
  - Choose indexes that can help many queries
  - Build indexes to support index-only strategies
  - Clustering is an important decision; only one index on a given relation can be clustered
  - Order of fields in composite index key can be important

Don’t Forget

- Homework #5 due this Wednesday (12/2)
- Homework #6 due on 12/11
- Project demo #2 this week
  - Expectation: almost done, close to what you will present to the whole class in final presentation
- Final: 12/16 8 - 9.50 a.m. @362 Willard