Introduction to Database Systems
CSE 444

Lecture 15: Data Storage and Indexes
Where We Are

• How to use a DBMS as a:
  – Data analyst: SQL, SQL, SQL,…
  – Application programmer: JDBC, XML,…
  – Database administrator: tuning, triggers, security
  – Massive-scale data analyst: Pig/MapReduce

• How DBMSs work:
  – Transactions
  – Data storage and indexing
  – Query execution

• Databases as a service
Outline

• Storage model

• Index structures (Section 14.1)

• B-trees (Section 14.2)
Storage Model

- DBMS needs spatial and temporal control over storage
  - Spatial control for performance
  - Temporal control for correctness and performance
    - Solution: Buffer manager inside DBMS (see past lectures)

- For spatial control, two alternatives
  - Use “raw” disk device interface directly
  - Use OS files
Spatial Control
Using “Raw” Disk Device Interface

• **Overview**
  – DBMS issues low-level storage requests directly to disk device

• **Advantages**
  – DBMS can ensure that important queries access data sequentially
  – Can provide highest performance

• **Disadvantages**
  – Requires devoting entire disks to the DBMS
  – Reduces portability as low-level disk interfaces are OS specific
  – Many devices are in fact “virtual disk devices”
Spatial Control
Using OS Files

• Overview
  – DBMS creates one or more very large OS files

• Advantages
  – Allocating large file on empty disk can yield good physical locality

• Disadvantages
  – OS can limit file size to a single disk
  – OS can limit the number of open file descriptors
  – But these drawbacks have mostly been overcome by modern OSs
Commercial Systems

• Most commercial systems offer both alternatives
  – Raw device interface for peak performance
  – OS files more commonly used

• In both cases, we end-up with a DBMS file abstraction implemented on top of OS files or raw device interface
Outline

• Storage model

• Index structures (Section 14.1)

• B-trees (Section 14.2)
  – [Old edition: 13.3]
Database File Types

The **data file** can be one of:

- **Heap file**
  - Set of records, partitioned into blocks
  - Unsorted

- **Sequential file**
  - Sorted according to some attribute(s) called **key**

“**key**” here means something else than “primary key”
Index

- A (possibly separate) file, that allows fast access to records in the data file
- The index contains (key, value) pairs:
  - The key = an attribute value
  - The value = one of:
    - pointer to the record (secondary index)
    - or the record itself (primary index)

“key” (aka “search key”) again means something else
Index Classification

• **Clustered/unclustered**
  – Clustered = data file is ordered by the index’s search key
  – Unclustered = otherwise

• **Primary/secondary**
  – Meaning 1: same as clustered/unclustered
  – Meaning 2:
    • Primary = index over set of fields that include the primary key
    • Secondary = not primary; index cannot reorder data, does not determine data location

• **Organization**: B+ tree or Hash table
Clustered Index

- File is sorted on the index attribute
- Only one per table
Unclustered Index

- Several per table

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>30</td>
</tr>
</tbody>
</table>
```
Clustered vs. Unclustered Index

CLUSTERED

B+ Tree

Data Records

UNCLUSTERED

B+ Tree

Data entries

(Index File)

(Data file)

Data Records
Outline

• Storage model

• Index structures (Section 14.1)

• B-trees (Section 14.2)
B+ Trees

• Search trees

• Idea in B Trees
  – Make 1 node = 1 block

• Idea in B+ Trees
  – Make leaves into a linked list: facilitates range queries
B+ Trees Basics

- Parameter $d = \text{the degree}$
- Each node has $\geq d$ and $\leq 2d$ keys (except root)

Each node also has $m+1$ pointers

Each leaf has $\geq d$ and $\leq 2d$ keys

CSE 444 - Summer 2010
B+ Tree Example

d = 2

Find the key 40
Using a B+ Tree

• Exact key values:
  – Start at the root
  – Proceed down, to the leaf

• Range queries:
  – As above
  – Then sequential traversal

Index on People(age)

Select name
From People
Where age = 25

Select name
From People
Where 20 <= age
and age <= 30
B+ Tree Design

• How large d?
• Example:
  – Key size = 4 bytes
  – Pointer size = 8 bytes
  – Block size = 4096 bytes
• $2d \times 4 + (2d+1) \times 8 \leq 4096$
• $d = 170$
B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%
  - average fanout = 133
- Typical capacities
  - Height 4: $133^4 = 312,900,700$ records
  - Height 3: $133^3 = 2,352,637$ records
- Can often hold top levels in buffer pool
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 Mbytes
Insertion in a B+ Tree

**Insert (K, P)**

- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:
  - If leaf, keep K3 too in right node
  - When root splits, new root has 1 key only
Insertion in a B+ Tree

Insert K=19
Insertion in a B+ Tree

After insertion
Insertion in a B+ Tree

Now insert 25
Insertion in a B+ Tree

After insertion
Insertion in a B+ Tree

But now have to split!
Insertion in a B+ Tree

After the split
Deletion from a B+ Tree

Delete 30
Deletion from a B+ Tree

After deleting 30

May change to 40, or not

CSE 444 - Summer 2010
Deletion from a B+ Tree

Now delete 25
Deletion from a B+ Tree

After deleting 25
Need to rebalance

Rotate
Deletion from a B+ Tree

Now delete 40
Deletion from a B+ Tree

After deleting 40
Rotation not possible
Need to **merge** nodes
Deletion from a B+ Tree

Final tree

CSE 444 - Summer 2010
Summary of B+ Trees

- Default index structure on most DBMS
- Very effective at answering ‘point’ queries: 
  `productName = 'gizmo'`
- Effective for range queries: 
  `50 < price AND price < 100`
- Less effective for multirange: 
  `50 < price < 100 AND 2 < quant < 20`