Where we stand

- How to use a DBMS as a:
  - Data analyst: SQL, SQL, SQL, ...
  - Application programmer: JDBC
  - Database admin: 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)
  - [Old edition: 13.1 and 13.2]

- B-trees (Section 14.2)
  - [Old edition: 13.3]
Memory hierarchy

Outline

- Storage model

- Index structures (Section 14.1)
  - [Old edition: 13.1 and 13.2]

- B-trees (Section 14.2)
  - [Old edition: 13.3]
High-level overview: Indexes

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
<th>salary</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>19</td>
<td>50k</td>
<td>...</td>
</tr>
<tr>
<td>005</td>
<td>20</td>
<td>55k</td>
<td>...</td>
</tr>
<tr>
<td>004</td>
<td>25</td>
<td>50k</td>
<td>...</td>
</tr>
<tr>
<td>007</td>
<td>30</td>
<td>80k</td>
<td>...</td>
</tr>
<tr>
<td>002</td>
<td>35</td>
<td>75k</td>
<td>...</td>
</tr>
<tr>
<td>003</td>
<td>35</td>
<td>70k</td>
<td>...</td>
</tr>
<tr>
<td>001</td>
<td>40</td>
<td>65k</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
<th>salary</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>006</td>
<td>19</td>
<td>50k</td>
<td>...</td>
</tr>
<tr>
<td>004</td>
<td>25</td>
<td>50k</td>
<td>...</td>
</tr>
<tr>
<td>005</td>
<td>20</td>
<td>55k</td>
<td>...</td>
</tr>
<tr>
<td>001</td>
<td>40</td>
<td>65k</td>
<td>...</td>
</tr>
<tr>
<td>007</td>
<td>30</td>
<td>80k</td>
<td>...</td>
</tr>
<tr>
<td>003</td>
<td>35</td>
<td>70k</td>
<td>...</td>
</tr>
<tr>
<td>002</td>
<td>35</td>
<td>75k</td>
<td>...</td>
</tr>
</tbody>
</table>

data file = index file
clustered (primary) index

index file
unclustered (secondary) index

http://www.cs.washington.edu/education/courses/cse444/11wi/
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 (sort) key

*different from "primary key"!
Index

- A (possibly separate) file, that allows fast access to records in the data file given a **search key**
- The index contains (key, value) pairs:
  - The key = an attribute value
  - The value = either a pointer to the record, or the record itself

again different from "primary key"!
Index Classification

- Clustered/unclustered
  - Clustered = records close in index are close in data
  - Unclustered = records close in index may be far in data

- Primary/secondary
  - Meaning 1: (Cow book)
    - Primary = is over attributes that include the primary key
    - Secondary = otherwise
  - Meaning 2: means the same as clustered/unclustered (Stanford book)

- Organization: B+ tree or Hash table

http://www.cs.washington.edu/education/courses/cse444/11wi/
Clustered/Unclustered

- **Clustered**
  - Index determines the location of indexed records
  - Typically, clustered index is one where values are data records (but not necessary)

- **Unclustered**
  - Index cannot reorder data, does not determine data location
  - In these indexes: value = pointer to data record
Clustered Index

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

- Several per table
Clustered vs. Unclustered Index

More commonly, in a clustered B+ Tree index, data entries are data records
Example hash-based index on sid (student id)

This is a primary index because it determines the order of indexed records

In this case, data entries in the index are actual data records
There is no separate data file

This index is also clustered
Hash-Based Index Example 2

h\text{2}(\text{age}) = 00

\begin{array}{c}
18 \\
18 \\
20 \\
22 \\
\end{array}

h\text{2}(\text{age}) = 01

\begin{array}{c}
19 \\
21 \\
21 \\
19 \\
\end{array}

\text{Secondary} index
Data entries in index are (key,RID) pairs

\text{Unclustered} index
Hash-Based Index

Good for point queries but not range queries

h2(age) = 00

h2(age) = 01

Another example of unclustered/secondary index

Another example of clustered/primary index

http://www.cs.washington.edu/education/courses/cse444/11wi/
Outline

- Storage model

- Index structures (Section 14.1)
  - [Old edition: 13.1 and 13.2]

- B-trees (Section 14.2)
  - [Old edition: 13.3]
B+ Trees

- Search trees

- Idea in B Trees
  - Make 1 node = 1 block
  - Keep tree balanced in height

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

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

Each node also has $m+1$ pointers

- Each leaf node has $d \leq m \leq 2d$ keys

Next leaf

Data records

http://www.cs.washington.edu/education/courses/cse444/11wi/
B+ Tree Example

Find the key 40

d = 2
B+ Tree Design

- How large d?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 bytes
- \(2d \times 4 + (2d+1) \times 8 \leq 4096\)
- \(d = 170\)
Searching a B+ Tree

- **Exact key values:**
  - Start at the root
  - Proceed down, to the leaf

- **Range queries:**
  - As above
  - Then sequential traversal

```sql
select name from people where age = 25
```

```sql
select name from people where 20 <= age and age <= 30
```
B+ Trees in Practice

- Typical degree: 100. Typical fill-factor: 67%
  - average fanout = 133

- Typical capacities
  - [Height 1: $133^1 = 133$ records]
  - Height 3: $133^3 = 2,352,637$ records
  - Height 4: $133^4 = 312,900,700$ 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

http://www.cs.washington.edu/education/courses/cse444/11wi/
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

$\text{d}=2$

http://www.cs.washington.edu/education/courses/cse444/11wi/
Insertion in a B+ Tree

After insertion

```
10 15 18 19
20 30 40 50
60 65
80 85 90
10 15 18 19
20 30 40 50
60 65
80 85 90
```

http://www.cs.washington.edu/education/courses/cse444/11wi/
Now insert 25
Insertion in a B+ Tree

After insertion

d=2
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

http://www.cs.washington.edu/education/courses/cse444/11wi/
Deletion from a B+ Tree

After deleting 30

May change to 40, or not
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 \textit{merge} nodes

http://www.cs.washington.edu/education/courses/cse444/11wi/
Deletion from a B+ Tree

Final tree
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

http://www.cs.washington.edu/education/courses/cse444/11wi/
Indexes in PostgreSQL

CREATE TABLE V(M int, N varchar(20), P int);

CREATE INDEX V1_N ON V(N);

CREATE INDEX V2 ON V(P, M);

CREATE INDEX VVV ON V(M, N);

CLUSTER V USING V2

Makes V2 clustered