ICS 321 Fall 2010
Overview of Storage & Indexing (i)

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Data Storage

- **Main Memory**
  - Random access
  - Volatile
- **Flash Memory**
  - Random access
  - Random writes are expensive
- **Disk**
  - Random access
  - Sequential access cheaper
- **Tapes**
  - Only sequential access
  - Archiving

Tertiary Storage
Relational Tables on Disk

- **Record** -- a tuple or row of a relational table
- **RIDs** – record identifiers that uniquely identify a record across memory and disk
- **Page** – a collection of records that is the unit of transfer between memory and disk
- **Bufferpool** – a piece of memory used to cache data and index pages.
- **Buffer Manager** – a component of a DBMS that manages the pages in memory
- **Disk Space Manager** – a component of a DBMS that manages pages on disk
Magnetic Disks

- A disk or platter contains multiple concentric rings called **tracks**.
- Tracks of a fixed diameter of a spindle of disks form a **cylinder**.
- Each track is divided into fixed sized **sectors** (i.e. “arcs”).
- Data stored in units of disk **blocks** (in multiples of sectors)
- An array of **disk heads** moves as a single unit.
- **Seek time**: time to move disk heads over the required track
- **Rotational delay**: time for desired sector to rotate under the disk head.
- **Transfer time**: time to actually read/write the data
Accessing Data on Disk

• **Seek time**: time to move disk heads over the required track

• **Rotational delay**: time for desired sector to rotate under the disk head.
  
  – Assume uniform distribution, on average time for half a rotation

• **Transfer time**: time to actually read/write the data
Example: Barracuda 1TB HDD (ST31000528AS)

• What is the average time to read 2048 bytes of data?
  
= Seek time + rotational latency + transfer time

= 8.5 msec + 4.16 msec + (2048 / 512) / 63 * (60000 msec / 7200 rpm)

= 8.5 + 4.16 + 0.265
File Organizations

How do we organize records in a file?

• **Heap files**: records not in any particular order
  – Good for scans

• **Sorted files**: records sorted by particular fields
  – Scans in the sorted order or range scans in the sorted order

• **Indexes**: Data structures to organize records via trees or hashing.
  – Like sorted files, they speed up searches for a subset of records, based on values in certain (“search key”) fields
  – Updates are much faster than in sorted files
Comparing File Organizations

Consider an employee table with search key <age,sal>

- **Scans**: fetch all records in the file
- **Point queries**: find all employees who are 30 years old (let’s assume there’s only one such employee)
- **Range queries**: find all employees aged above 65.
- **Insert** a record.
- **Delete** a record given its RID.
Analysis of Algorithms

• Computation model
  – CPU comparison operation
  – General: most expensive operation
• Worst-case
  – How bad can it get?
• Average-case
  – Assumption about probabilities
• Analysis: count the number of some operation w.r.t. some input size
• Asymptotics: Big “O”
  – Constants don’t matter
  – 500n+10000 = O(n)

SELECT *
FROM Employees E
WHERE E.age=30

For each tuple t in Employees
{
  if (t.age==30)
  {
    output t
  }
}

What is the worse case number of output tuples?

What is the worse case running time in the number of comparisons?

Assume input size : n tuples

11/10/2010
Lipyeow Lim -- University of Hawaii at Manoa
SELECT * 
FROM Employees E 
WHERE E.age=30

Tuples are sorted by age

Shortcircuited Linear Search

For each tuple t in Employees
{
    if (t.age==30)
    {
        output t
    }
    elseif ( t.age > 30 )
    {
        exit
    }
}

Binary Search

(lo, hi) = (0,n-1) 
mid = lo+(hi-lo)/2
While(hi>lo & & E[mid].age!=30)
{
    if (E[mid].age < 30)
    {
        lo=mid
    }
    else
    {
        hi=mid
    }
    mid = lo+(hi-lo)/2
}
Output all satisfying tuples around E[mid]

What is the worse case running time in the number of comparisons ?
Analysis of Binary Search

- Number tuples searched per iteration = n, n/2, n/4, ... 1
- Hence the number of iterations = O( \log n )
- Therefore number of comparisons = O(\log n)

What is the worse case?

(lo, hi) = (0, n-1)
mid = lo + (hi-lo)/2
While(hi>lo && E[mid].age!=30) {
    if (E[mid].age < 30) {
        lo=mid
    }
    else {
        hi=mid
    }
    mid = lo + (hi-lo)/2
}
Output all satisfying tuples around E[mid]
SELECT *
FROM Employees
WHERE age=30

for each page \( p \) of Employees table
{
    if \((p \text{ not in bufferpool})\)
    {
        Fetch \( p \) from disk
    }
    for each tuple \( t \) in page \( p \)
    {
        if \((t\.age==40)\)
        {
            output \( t \)
        }
    }
}

Table Scan

Worst case running time =
+ \text{time to fetch all pages of Movies from disk}
+ \text{time to compare studioNames}
+ \text{time to output result}

How would you estimate these times?

What is the worst case number of disk access?

What is the most expensive operation?
Analysis Model

• $B$ : number of data pages
• $R$ : number of records per page
• $D$ : average time to read/write a disk page
  – From previous calculations, if a page is 2K bytes, $D$ is about 13 milliseconds
• $C$ : average time to process a record
  – For the 1 Ghz processors we have today, assuming it takes 100 cycles, $C$ is about 100 nanoseconds
Table Scans on Heap Files

- SELECT * FROM Employees
  - O(B) pages get fetched + O(B*R) tuples processed

- SELECT * FROM Employees
  WHERE age=30

- SELECT * FROM Employees
  WHERE age > 20 and age < 30

for each page \( p \) of Employees table
{
  if (\( p \) not in bufferpool)
  {
    Fetch \( p \) from disk
  }
  for each tuple \( t \) in page \( p \)
  {
    output \( t \)
    if (\( t \).age==30)
    {
      output \( t \)
    }
    if (\( t \).age>20 && \( t \).age<30)
    {
      output \( t \)
    }
  }
}
## Analysis of Heap File Storage

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst Case Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scans</td>
<td>( B^* (D + R*C) )</td>
</tr>
<tr>
<td>Point Query</td>
<td>( B^* (D + R*C) )</td>
</tr>
<tr>
<td>Range Query</td>
<td>( B^* (D + R*C) )</td>
</tr>
<tr>
<td>Insert</td>
<td>( 2*D + C )</td>
</tr>
<tr>
<td>Delete</td>
<td>( 2<em>B</em>(D + R*C) )</td>
</tr>
</tbody>
</table>

- **Fetch all** \( B \) **pages from disk into memory**
- **Process each record on each page**
- **In the worst case, the desired record is the last record on the last page**
- **Since file is unsorted, the desired records can be anywhere in the file, so we have to scan the entire file.**
- **Insert at the end of the file.**
- **Read in the last page**
- **Add record**
- **Write the page back**
- **Search for the record to be deleted**
- **Delete the record**
- **Move all subsequent records & pages forward.**
## Analysis of Heap File Storage (Disk Only)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst Case Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scans</td>
<td>B*D</td>
</tr>
<tr>
<td>Point Query</td>
<td>B*D</td>
</tr>
<tr>
<td>Range Query</td>
<td>B*D</td>
</tr>
<tr>
<td>Insert</td>
<td>2*D</td>
</tr>
<tr>
<td>Delete</td>
<td>2<em>B</em>D</td>
</tr>
</tbody>
</table>

**Operation Analysis:**
- **Scans**
  - Fetch all B pages from disk into memory
  - Process each record on each page
- **Point Query**
  - In the worst case, the desired record is the last record on the last page
- **Range Query**
  - Since file is unsorted, the desired records can be anywhere in the file, so we have to scan the entire file.
- **Insert**
  - Insert at the end of the file.
  - Read in the last page
  - Add record
  - Write the page back
- **Delete**
  - Search for the record to be deleted
  - Delete the record
  - Move all subsequent records & pages forward.
Deleting a Record

File

Memory

Record to be deleted
## Analysis of Sorted File Storage

<table>
<thead>
<tr>
<th>Op</th>
<th>Worst Case Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scans</td>
<td>$B^* (D + R^*C)$</td>
</tr>
<tr>
<td>Point Query</td>
<td>$D \log B + C \log R$</td>
</tr>
<tr>
<td>Range Query</td>
<td>$D \log B + C \log R$ + $\lfloor S/R \rfloor * D + S^*C$</td>
</tr>
<tr>
<td>Insert</td>
<td>$D \log B + C \log R$ + $2B^*(D + R^*C)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$D \log B + C \log R$ + $2B^*(D + R^*C)$</td>
</tr>
</tbody>
</table>

- **Scans**
  - Fetch all $B$ pages from disk into memory
  - Process each record on each page

- **Point Query**
  - Binary search for the desired page
  - Binary search for the desired record within the page

- **Range Query**
  - Let $S$ be the number of records in the result
  - Binary search for the desired page and record
  - Fetch the next $S$ records
  - Binary search to insertion point
  - In worst case, page has no extra space, so page is split
  - Move all subsequent pages back

- **Insert**
  - Search for the record to be deleted
  - Delete the record
  - Move all subsequent pages forward
## Heap vs Sorted File

<table>
<thead>
<tr>
<th>Op</th>
<th>Heap</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scans</td>
<td>$B^*D$</td>
<td>$B^*D$</td>
</tr>
<tr>
<td>Point Query</td>
<td>$B^*D$</td>
<td>$D \log B$</td>
</tr>
<tr>
<td>Range Query</td>
<td>$B^*D$</td>
<td>$D \log B + \lceil S/R \rceil * D$</td>
</tr>
<tr>
<td>Insert</td>
<td>$2^*D$</td>
<td>$D \log B + 2^B*D$</td>
</tr>
<tr>
<td>Delete</td>
<td>$2^B^*D$</td>
<td>$D \log B + 2^<em>B</em>D$</td>
</tr>
</tbody>
</table>