Database Systems
CSE 414

Lectures 11 – 12:
Basics of Query Optimization and Cost Estimation
(Ch. 15.{1,3,4,6,6} & 16.4-5)
Announcements

• HW3 is due Tuesday

• WQ4 is due Thursday

• Midterm on Friday
  – we’ll talk more about it on Monday

• Husky Football spring game tomorrow
Motivation

• To understand performance, need to understand a bit about how a DBMS works
  – my database application is too slow… why?
  – one of the queries is very slow… why?

• Under your direct control: index choice
  – understand how that affects query performance
Recap: Query Evaluation

1. SQL query
   - Parse & Check Query
     - Translate query string into internal representation
     - Check syntax, access control, table names, etc.
   - Decide how best to answer query: query optimization
2. Query Execution
3. Return Results
Query Optimizer Overview

- **Input**: Parsed & checked SQL
- **Output**: A good physical query plan
- **Basic query optimization algorithm**:
  - Enumerate alternative plans (logical and physical)
  - Compute estimated cost of each plan
    - Compute number of I/Os
    - Optionally take into account other resources
  - Choose plan with lowest cost
  - This is called cost-based optimization
Query Optimizer Overview

• There are exponentially many query plans
  – exponential in the size of the query
  – simple SFW with 3 joins has not too many
• Optimizer will consider many, many of them
• Worth substantial cost to avoid bad plans
Rest of Today

- Cost of reading from disk
- Cost of single RA operators
- Cost of query plans
Cost of Reading Data From Disk
Cost Parameters

- Cost = Disk I/O + CPU + Network I/O
  - We will focus on Disk I/O

- Parameters:
  - \( B(R) \) = # of blocks (i.e., pages) for relation R
  - \( T(R) \) = # of tuples in relation R
  - \( V(R, A) \) = # of distinct values of attribute a
    - When \( A \) is a key, \( V(R,A) = T(R) \)
    - When \( A \) is not a key, \( V(R,A) \) can be anything < \( T(R) \)

- Where do these values come from?
  - DBMS collects statistics about data on disk
Selectivity Factors for Conditions

- **A = c** /* σ_{A=c}(R) */
  - Selectivity = \(1/V(R,A)\)

- **A < c** /* σ_{A<c}(R)*/
  - Selectivity = \((c - \text{Low}(R, A))/(\text{High}(R,A) - \text{Low}(R,A))\)

- **c1 < A < c2** /* σ_{c1<A<c2}(R)*/
  - Selectivity = \((c2 – c1)/(\text{High}(R,A) - \text{Low}(R,A))\)
Example: Selectivity of $\sigma_{A=c}(R)$

<table>
<thead>
<tr>
<th>$T(R)$</th>
<th>100,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V(R, A)$</td>
<td>20</td>
</tr>
</tbody>
</table>

How many records are returned by $\sigma_{A=c}(R) =$ ?

Answer: $X \times T(R)$, where $X = \text{selectivity}$…

... $X = 1/V(R,A) = 1/20$

Number of records returned = $100,000/20 = 5,000$
Cost of Index-based Selection

• Sequential scan for relation R costs $B(R)$

• Index-based selection
  – Estimate selectivity factor $X$ (see previous slide)
  – Clustered index: $X*B(R)$
  – Unclustered index $X*T(R)$

Note: we are ignoring I/O cost for index pages
Example: Cost of $\sigma_{A=c}(R)$

- Example:
  \[
  \begin{array}{l}
  B(R) = 2000 \\
  T(R) = 100,000 \\
  V(R, A) = 20
  \end{array}
  \]
  - cost of $\sigma_{A=c}(R) = \?$

- Table scan: $B(R) = 2,000$ I/Os

- Index based selection:
  - If index is clustered: $B(R)/V(R,A) = 100$ I/Os
  - If index is unclustered: $T(R)/V(R,A) = 5,000$ I/Os

Lesson: Don’t build unclustered indexes when $V(R,A)$ is small!
Cost of Executing Operators
(Focus on Joins)
Outline

• Join operator algorithms
  – One-pass algorithms (Sec. 15.2 and 15.3)
  – Index-based algorithms (Sec 15.6)

• Note about readings:
  – In class, we discuss only algorithms for joins
  – Other operators are easier: read the book
Join Algorithms

- Hash join
- Nested loop join
- Sort-merge join
Hash Join

Hash join: $R \bowtie S$

- Scan $R$, build buckets in main memory
- Then scan $S$ and join
- Cost: $B(R) + B(S)$

- One-pass algorithm when $B(R) \leq M$
  - more disk access also when $B(R) > M$
Hash Join Example

Patient(pid, name, address)
Insurance(pid, provider, policy_nb)
Patient \( \bowtie \) Insurance

<table>
<thead>
<tr>
<th></th>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>‘Bob’</td>
<td>‘Blue’ 123</td>
</tr>
<tr>
<td>2</td>
<td>‘Ela’</td>
<td>‘Prem’ 432</td>
</tr>
<tr>
<td>3</td>
<td>‘Jill’</td>
<td>‘Prem’ 343</td>
</tr>
<tr>
<td>4</td>
<td>‘Joe’</td>
<td>‘GrpH’ 554</td>
</tr>
<tr>
<td></td>
<td>‘Seattle’</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>‘Everett’</td>
<td></td>
</tr>
</tbody>
</table>

Two tuples per page
Hash Join Example

Patient $\bowtie$ Insurance

Showing pid only

Memory M = 21 pages

Disk

Patient | Insurance
---|---
1 2 | 2 4 | 6 6
3 4 | 4 3 | 1 3
9 6 | 2 8 |
8 5 | 8 9 |

Large enough

This is one page with two tuples
Hash Join Example

Step 1: Scan Patient and build hash table in memory

Memory M = 21 pages

Hash h: pid % 5

Disk

Patient | Insurance
---|---
1 | 2
3 | 4
9 | 6
8 | 5

Input buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory M = 21 pages

Hash h: pid % 5

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8 8 9</td>
</tr>
<tr>
<td>8 5</td>
<td></td>
</tr>
</tbody>
</table>

Input buffer

Output buffer

Write to disk
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory M = 21 pages

Hash h: pid % 5

Disk

Patient  Insurance

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input buffer

Output buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory $M = 21$ pages

Hash $h$: pid $\% 5$

Input buffer

Output buffer

Keep going until read all of Insurance

Cost: $B(R) + B(S)$
Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- $R$ is the outer relation, $S$ is the inner relation

```plaintext
for each tuple $t_1$ in $R$ do
  for each tuple $t_2$ in $S$ do
    if $t_1$ and $t_2$ join then output $(t_1, t_2)$
```

- **Cost**: $B(R) + T(R)B(S)$
- Multiple-pass since $S$ is read many times
Block-at-a-time Refinement

for each block of tuples r in R do
  for each block of tuples s in S do
    for all pairs of tuples t₁ in r, t₂ in s
      if t₁ and t₂ join then output (t₁,t₂)

• Cost: B(R) + B(R)B(S)
Block-at-a-time Refinement

Disk

Patient  Insurance
1  2  2  4  6  6
3  4  4  3  1  3
9  6  2  8
8  5  8  9

Input buffer for Patient
1  2
2  4

Input buffer for Insurance
Output buffer
2  2
Block-at-a-time Refinement

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Input buffer for Patient

Input buffer for Insurance

Output buffer
Page-at-a-time Refinement

Disk

Patient

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Insurance

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Input buffer for Patient

Input buffer for Insurance

Keep going until read all of Insurance

Output buffer
Block-at-a-time Refinement

Cost: $B(R) + B(R)B(S)$
Block-Nested-Loop Refinement

\[
\text{for each group of M}-1 \text{ pages } r \text{ in } R \text{ do} \\
\hspace{1em} \text{for each page of tuples } s \text{ in } S \text{ do} \\
\hspace{2em} \text{for all pairs of tuples } t_1 \text{ in } r, t_2 \text{ in } s \\
\hspace{3em} \text{if } t_1 \text{ and } t_2 \text{ join then output } (t_1, t_2)
\]

- Cost: $B(R) + \frac{B(R)B(S)}{(M-1)}$
Sort-Merge Join

Sort-merge join:  \( R \Join S \)

- Scan \( R \) and sort in main memory
- Scan \( S \) and sort in main memory
- Merge \( R \) and \( S \)

- Cost: \( B(R) + B(S) \)
- One pass algorithm when \( B(S) + B(R) \leq M \)
- Typically, this is NOT a one pass algorithm
Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

Memory M = 21 pages

Disk

Patient  Insurance

1 2  2 4  6 6
3 4  4 3  1 3
9 6  2 8  8 9
**Sort-Merge Join Example**

**Step 2: Scan Insurance and sort in memory**

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

**Memory M = 21 pages**

```
1 2 3 4 5 6 8 9
1 2 2 3 3 4 4 6
6 8 8 9
```
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory M = 21 pages
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Disk

Patient  Insurance

1 2  2 4  6 6
3 4  4 3  1 3
9 6  2 8
8 5  8 9

Output buffer

2 2
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Using PK, so only one can match

Output buffer

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 3 3 4 4 6</td>
</tr>
<tr>
<td>6 8 8 9</td>
</tr>
<tr>
<td>2 2</td>
</tr>
</tbody>
</table>
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Disk

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Memory M = 21 pages

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 3 3 4 4 6</td>
</tr>
<tr>
<td>6 8 8 9</td>
</tr>
<tr>
<td>3 3</td>
</tr>
</tbody>
</table>

Output buffer
Sort-Merge Join Example

Step 3: **Merge** Patient and Insurance

<table>
<thead>
<tr>
<th>Disk</th>
<th>Memory M = 21 pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient</td>
<td>Insurance</td>
</tr>
<tr>
<td>1 2</td>
<td>2 4 6 6</td>
</tr>
<tr>
<td>3 4</td>
<td>4 3 1 3</td>
</tr>
<tr>
<td>9 6</td>
<td>2 8</td>
</tr>
<tr>
<td>8 5</td>
<td>8 9</td>
</tr>
</tbody>
</table>

Output buffer

Keep going until end of first relation
Index Nested Loop Join

\( R \bowtie S \)

- Assume \( S \) has an index on the join attribute
- Iterate over \( R \), for each tuple fetch corresponding tuple(s) from \( S \)

- **Cost:**
  - If index on \( S \) is clustered: \( B(R) + T(R)B(S)/V(S,A) \)
  - If index on \( S \) is unclustered: \( B(R) + T(R)T(S)/V(S,A) \)
Cost of Query Plans
Physical Query Plan 1

(On the fly) \[ \pi_{\text{sname}} \] Selection and project on-the-fly
-> No additional cost.

(On the fly) \[ \sigma_{\text{scity}='Seattle' \land \text{sstate}='WA' \land \text{pno}=2} \]

(Nested loop) \[ \text{sno} = \text{sno} \]

Total cost of plan is thus cost of join:
= \[ B(\text{Supplier}) + B(\text{Supplier}) \times B(\text{Supply}) \]
= \[ 100 + 100 \times 100 \]
= 10,100 I/Os

T(\text{Supplier}) = 1000
T(\text{Supply}) = 10,000
B(\text{Supplier}) = 100
B(\text{Supply}) = 100
V(\text{Supplier,scity}) = 20
V(\text{Supplier,state}) = 10
V(\text{Supply,pno}) = 2,500
M = 11
Physical Query Plan 2

(On the fly) $\pi_{\text{aname}}$

(Sort-merge join) $\text{sno} = \text{sno}$

(Scan write to T1) $\sigma_{\text{scity}='Seattle' \land \text{state}='WA'}$

(Scan write to T2) $\sigma_{\text{pno}=2}$

(a) $\sigma_{\text{scity}='Seattle' \land \text{state}='WA'}$

(b) $\sigma_{\text{pno}=2}$

Supplier (File scan)

Supply (File scan)

Total cost

\[= 100 + 100 \times \frac{1}{20} \times \frac{1}{10} \text{ (a)} + 100 + 100 \times \frac{1}{2500} \text{ (b)} + 2 \text{ (c)} + 0 \text{ (d)}\]

Total cost \(\approx 204 \text{ I/Os}\)

$T(\text{Supplier}) = 1000$

$B(\text{Supplier}) = 100$

$V(\text{Supplier}, \text{scity}) = 20$

$M = 11$

$T(\text{Supply}) = 10,000$

$B(\text{Supply}) = 100$

$V(\text{Supplier}, \text{state}) = 10$

$V(\text{Supply}, \text{pno}) = 2,500$
Physical Query Plan 3

(On the fly)  (d) \( \pi \text{sname} \)

(On the fly)

(On the fly)

(Use hash index)

(Use hash index)

(On the fly)

Total cost

\[= 1 \text{ (a)} + 4 \text{ (b)} + 0 \text{ (c)} + 0 \text{ (d)} \]

Total cost \(\approx 5 \text{ I/Os} \)

\(\sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'}\)

\(\sigma_{\text{pno}=2}\)

Assume: clustered

(Use hash index)

4 tuples

((Index nested loop)

(M = 11)

Components:

\(\text{T(Supplier)} = 1000\)

\(\text{B(Supplier)} = 100\)

\(\text{V(Supplier, scity)} = 20\)

\(\text{B(Supplier)} = 100\)

\(\text{V(Supplier, state)} = 10\)

\(\text{V(Supply, pno)} = 2,500\)

\(\text{T(Supply)} = 10,000\)

\(\text{V(Supply, pno)} = 2,500\)

\(\text{V(Supplier, scity)} = 20\)

\(\text{V(Supplier, state)} = 10\)

\(\text{M} = 11\)