Database Systems
CSE 344

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

• HW3 is due next Tuesday
  – Azure setup can take awhile. Get this done by Friday!

• Midterm next Friday
  – we’ll talk more about it on Monday
Recap: Query Evaluation

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

Logical plan → physical plan

Query Execution

Return Results

Query Evaluation

CSE 344 - Summer 2017 3
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**
Index Classification

• **Clustered/unclustered**
  – Clustered = records close in index are close in data
    • Option 1: Data inside data file is sorted on disk
    • Option 2: Store data directly inside the index (no separate files)
  – Unclustered = records close in index may be far in data

• **Primary/secondary**
  – Meaning 1:
    • Primary = is over attributes that include the primary key
    • Secondary = otherwise
  – Meaning 2: means the same as clustered/unclustered

• **Organization** $B^+$ tree or Hash table
Basic Index Selection Guidelines

• Consider queries in workload in order of importance
  – ignore infrequent queries if you also have many writes

• Consider relations accessed by query
  – No point indexing other relations

• Look at WHERE clause for possible search key

• Try to choose indexes that speed-up multiple queries
To Cluster or Not

• Range queries benefit mostly from clustering

• **Covering indexes do not** need to be clustered: they work equally well unclustered
  – (a covering index for a query is one where every attribute mentioned in the query is part of the index’s search key)
  – in that case, index has all the info you need anyway

\[ \text{Index} \text{ } V \{a, b, c\} \]
\[ \text{select } a, b, c, d \]
The query returns only a few records

The query returns almost all records in R

SELECT * FROM R WHERE K>? and K<?

Cost

Percentage tuples retrieved

CSE 344 - Summer 2017
SELECT *  
FROM R  
WHERE K>? and K<?
Clustered vs Unclustered

Every table can have **only one** clustered and **many** unclustered indexes.

SQL Server defaults to cluster by **primary key**.
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:
  – $B(R) = \# \text{ of blocks (i.e., pages) for relation } R$
  – $T(R) = \# \text{ of tuples in relation } R$
  – $V(R, A) = \# \text{ of distinct values of attribute } a$
    • 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**  
  - Selectivity = \(1/V(R,A)\)

- **A < c**  
  - Selectivity = \((c - \min(R,A))/\left(\max(R,A) - \min(R,A)\right)\)

- **c1 < A < c2**  
  - Selectivity = \((c2 - c1)/\left(\max(R,A) - \min(R,A)\right)\)

Assume uniform distribution
Example: Selectivity of $\sigma_{A=c}(R)$

\[
\begin{align*}
T(R) &= 100,000 \\
V(R, A) &= 20
\end{align*}
\]

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:
  
  \[
  \begin{align*}
  B(R) &= 2000 \\
  T(R) &= 100,000 \\
  V(R, A) &= 20
  \end{align*}
  \]

  \text{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 \Join Insurance

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Bob'</td>
</tr>
<tr>
<td>2</td>
<td>'Ela'</td>
</tr>
<tr>
<td>3</td>
<td>'Jill'</td>
</tr>
<tr>
<td>4</td>
<td>'Joe'</td>
</tr>
</tbody>
</table>

Two tuples per page
Hash Join Example

Patient \Join Insurance

Disk

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

Memory M = 21 pages

Showing pid only

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

Input buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory $M = 21$ pages

Hash $h: pid \mod 5$

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

Disk

Patient Insurance

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

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 $\% 5$

Input buffer

Output buffer
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Memory $M = 21$ pages

Hash $h: \text{pid} \mod 5$

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

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

\[
\text{for each tuple } t_1 \text{ in } R \text{ do }
\text{for each tuple } t_2 \text{ in } S \text{ do }
\text{if } t_1 \text{ and } t_2 \text{ join then output } (t_1, t_2)
\]

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

\[
\text{Cost: } B(R) + B(R)B(S)
\]

for each block of tuples r in R do
    for each block of tuples s in S do
        for all pairs of tuples \(t_1\) in r, \(t_2\) in s
            if \(t_1\) and \(t_2\) join then output \((t_1,t_2)\)

What is the Cost?
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

Input buffer for Insurance

Output buffer

CSE 344 - Summer 2017
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

Input buffer for Insurance

Output buffer

CSE 344 - Summer 2017
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 |

Input buffer for Patient
Input buffer for Insurance

Keep going until read all of Insurance

Output buffer

CSE 344 - Summer 2017
**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</td>
</tr>
<tr>
<td>3 4</td>
<td>4 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

Keep going until read all of Insurance

Then repeat for next page of Patient… until end of Patient

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

When both relations don’t fit into memory

```plaintext
for each group of M-1 blocks 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: \( B(R) + \frac{B(R)B(S)}{(M-1)} \)
Sort-Merge Join

Sort-merge join:  $R \bowtie S$

- 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

<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>
Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

Disk

Patient  Insurance

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

Memory M = 21 pages

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

Output buffer

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

Output buffer

| 2 2 |

CSE 344 - Summer 2017
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

Memory $M = 21$ pages

Using PK, so only one can match
### Sort-Merge Join Example

**Step 3: Merge Patient and Insurance**

- **Disk**
  - **Patient**: 1 2 3 4
  - **Insurance**: 5 6 8 9

- **Memory M = 21 pages**
  - 1 2 3 4 5 6 8 9
  - 1 2 3 4 6
  - 3 3

- **Output buffer**

---

CSE 344 - Summer 2017
Sort-Merge Join Example

Step 3: **Merge** Patient and Insurance

Memory $M = 21$ pages

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: \( B(R) + T(R)T(S)/V(S,A) \)

15.6.3
Cost of Query Plans
Physical Query Plan 1

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

\( \sigma \text{ scity='Seattle' } \wedge \text{sstate='WA' } \wedge \text{pno=2} \)  
Total cost of plan is thus cost of join:  
= \( B(\text{Supplier}) + B(\text{Supplier}) \cdot B(\text{Supply}) \)  
= 100 + 100 * 100  
= 10,100 I/Os

\( \text{T(Supplier)} = 1000 \)  
\( \text{B(Supplier)} = 100 \)  
\( \text{V(Supplier,scity)} = 20 \)  
\( \text{M} = 11 \)

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

(a) $\sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'}$
(b) $\sigma_{\text{pno}=2}$
(c) $\pi_{\text{sname}}$
(d) Path selection down

Total cost:
- $100 + 100 \times 1/20 \times 1/10$ (a)
- $100 + 100 \times 1/2500$ (b)
- $2$ (c)
- $0$ (d)

Total cost $\approx 204$ I/Os

B(Supplier) = 100
B(Supply) = 100
V(Supplier,scity) = 20
V(Supplier,sstate) = 10
V(Supply,pno) = 2,500
M = 11
(On the fly)  

(b) \( \sigma_{sno = sno} \)  

(Use hash index)  

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

(On the fly)  

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

(On the fly)  

(d) \( \pi_{\text{sname}} \)  

Using Index  

Physical Query Plan 3  

Total cost \( \approx 5 \) I/Os

\( \sigma_{sno = sno} \)  

\( \pi_{\text{sname}} \)  

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

\( \sigma_{pno = 2} \)  

Using Index

Total cost  

= 1 (a)  

+ 4 (b)  

+ 0 (c)  

+ 0 (d)  

\( \approx 5 \) I/Os

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

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

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

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

\( V(\text{Supply}, \text{pno}) = 2500 \)  

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

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

\( V(\text{Supply}, \text{pno}) = 2500 \)  

\( M = 11 \)