CSE 344

APRIL 27TH – COST ESTIMATION
ADMINISTRIVIA

• HW5 Out
  • Please verify that you can run queries
• Midterm
  • May 9\textsuperscript{th} 9:30-10:20 – MLR 301
  • Review (in class) – May 7\textsuperscript{th}
  • Practice exam – May 4\textsuperscript{th}
• Through parallelism: next week’s material
INDEX BASED SELECTION

Example:

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

cost of \( \sigma_{a=v}(R) = ? \)

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

Index based selection:

- If index is clustered: \( B(R) \times 1/V(R,a) = 100 \) I/Os
- If index is unclustered: \( T(R) \times 1/V(R,a) = 5,000 \) I/Os

Lesson: Don’t build unclustered indexes when \( V(R,a) \) is small!
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: \( B(R) + B(S) \)
Which relation to build the hash table on?
HASH JOIN

Hash join:  \( R \bowtie S \)
Scan \( R \), build buckets in main memory
Then scan \( S \) and join
Cost: \( B(R) + B(S) \)
Which relation to build the hash table on?

One-pass algorithm when \( B(R) \leq M \)
- \( M = \) number of memory pages available
HASH JOIN EXAMPLE

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

Patient \( \bowtie \) Insurance

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>'Bob'</th>
<th>'Seattle'</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>'Ela'</td>
<td>'Everett'</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>'Jill'</td>
<td>'Kent'</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>'Joe'</td>
<td>'Seattle'</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>'Blue'</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>'Prem'</td>
<td>432</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>'Prem'</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>'GrpH'</td>
<td>554</td>
<td></td>
</tr>
</tbody>
</table>

Two tuples per page
**HASH JOIN EXAMPLE**

Patient \( \bowtie \) Insurance

- Memory \( M = 21 \) pages

**Showing pid only**

- This is one page with two tuples

- Some large-enough #

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>
HASH JOIN EXAMPLE

Step 1: Scan Patient and **build** hash table in memory
Can be done in method open()

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

| 5 | 1 | 6 | 2 | 3 8 4 9 |
**HASH JOIN EXAMPLE**

Step 2: Scan Insurance and **probe** into hash table

Done during calls to next()

Memory $M = 21$ pages

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

<table>
<thead>
<tr>
<th>Disk</th>
<th>Patient</th>
<th>Insurance</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>

| 2 4 | 6 6 | 1 3 | 1 6 |

Input buffer

2 4

Output buffer

2 2

Write to disk or pass to next operator
HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table
Done during calls to next()

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

Memory M = 21 pages
Hash h: pid % 5

Disk

Input buffer

Output buffer
HASH JOIN EXAMPLE

Step 2: Scan Insurance and probe into hash table
Done during calls to next()

Memory M = 21 pages
Hash h: pid % 5

Disk

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

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 \((t_1,t_2)\)
```

What is the Cost?
NESTED LOOP JOINS

Tuple-based nested loop $R \bowtie S$

$R$ is the outer relation, $S$ is the inner relation

```
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: $B(R) + T(R) \cdot B(S)$

Multiple-pass since $S$ is read many times
PAGE-AT-A-TIME REFINEMENT

for each page of tuples $r$ in $R$ do
  for each page 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 ($t_1$, $t_2$)

Cost: $B(R) + B(R)B(S)$
PAGE-AT-A-TIME REFINEMENT

Disk

Patient   Insurance

1 2
3 4
9 6
8 5

Input buffer for Patient

1 2
2 4

Input buffer for Insurance

6 6
1 3

Output buffer

8 9
PAGE-AT-A-TIME REFINEMENT

Disk

Patient  Insurance

1  2
3  4
9  6
8  5

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

Input buffer for Patient
4  3
Input buffer for Insurance

Output buffer
PAGE-AT-A-TIME REFINEMENT

Disk

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

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

for each group of M-1 pages r in R do
  for each page 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 ($t_1$, $t_2$)

Cost: $B(R) + B(R)B(S)/(M-1)$
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

<table>
<thead>
<tr>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<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>

<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>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
**SORT-MERGE JOIN EXAMPLE**

Step 2: Scan Insurance 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>

```plaintext
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

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>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Output buffer

| 1 | 1 |
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:
  \[ B(R) + T(R) \times (B(S) \times \frac{1}{V(S,a)}) \]
- If index on S is unclustered:
  \[ B(R) + T(R) \times (T(S) \times \frac{1}{V(S,a)}) \]
LOGICAL QUERY PLAN 1

\[ \sigma_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \Pi_{\text{sname}} \]

SELECT \text{sname} 
FROM \text{Supplier} \text{x, Supply} \text{y} 
WHERE \text{x.sid} = \text{y.sid} 
and \text{y.pno} = 2 
and \text{x.scity} = 'Seattle' 
and \text{x.sstate} = 'WA'

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply, pno}) = 2500 \]

\[ T(\text{Supplier}) = 1000 \]
\[ B(\text{Supplier}) = 100 \]
\[ V(\text{Supplier, scity}) = 20 \]
\[ V(\text{Supplier, state}) = 10 \]

\[ M=11 \]
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11
LOGICAL QUERY PLAN 1

\[
\text{SELECT sname FROM Supplier x, Supply y WHERE } x.\text{sid} = y.\text{sid} \land y.pno = 2 \\
\text{and } x.\text{scity} = \text{'Seattle'} \land x.\text{sstate} = \text{'WA'}
\]

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10

M = 11

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
LOGICAL QUERY PLAN 2

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'
```

**Table and Relational Algebra**

- **Table: Supplier**
  - `sid` (Supplier ID)
  - `sname` (Supplier Name)
  - `scity` (City)
  - `sstate` (State)

- **Table: Supply**
  - `sid` (Supplier ID)
  - `pno` (Part Number)
  - `quantity`

**Indexes and Statistics**

- `T(Supplier) = 1000`
- `B(Supplier) = 100`
- `V(Supplier, scity) = 20`
- `V(Supplier, sstate) = 10`
- `T(Supply) = 10000`
- `B(Supply) = 100`
- `V(Supply, pno) = 2500`

**M=11**
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

LOGICAL QUERY PLAN 2

Very wrong! Why?

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
M=11
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

M = 11
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
LOGICAL QUERY PLAN 2

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, sstate) = 10

M=11
PHYSICAL PLAN 1

\[ \text{σ}_{\text{pno}=2 \land \text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \text{T} = 10000 \]

\[ \text{sid} = \text{sid} \]

Block nested loop join

Scan

Supply

Scan

Supplier

Total cost:

\[ \text{M=11} \]

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]

\[ \text{T(Supplier)} = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]
PHYSICAL PLAN 1

\[
\sigma_{pno=2 \land scity='Seattle' \land sstate='WA'}
\]

\[
\pi_{sname} \quad T(\text{Supplier}) = 1000
\]

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

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

\[
V(\text{Supplier}, \text{sstate}) = 10
\]

\[
T(\text{Supply}) = 10000
\]

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

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

\[
\text{Total cost: } 100 + 100 \times 100 / 10 = 1100
\]

Cost: \( B(R) + B(R)B(S)/(M-1) \)

\( M=11 \)
PHYSICAL PLAN 2

\[ \text{Cost of Supply}(pno) = 4 \]
\[ \text{Cost of Supplier}(scity) = 50 \]
\[ \text{Total cost:} 54 \]

\[ T(\text{Supply}) = 10000 \]
\[ B(\text{Supply}) = 100 \]
\[ V(\text{Supply, pno}) = 2500 \]

\[ T(\text{Supplier}) = 1000 \]
\[ B(\text{Supplier}) = 100 \]
\[ V(\text{Supplier, scity}) = 20 \]
\[ V(\text{Supplier, state}) = 10 \]

\[ M = 11 \]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

**PHYSICAL PLAN 2**

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54

Unclustered index lookup Supplier(scity)

Unclustered index lookup Supply(pno)

Main memory join

\[ \Pi_{\text{sname}} \]

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

\( \sigma_{\text{state}='WA'} \)

\( \sigma_{\text{scity}='Seattle'} \)

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

M=11
 PHYSICAL PLAN 2

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

**PHYSICAL PLAN 3**

\[ \pi_{\text{sname}}(\text{Supplier}) \]

\[ \sigma_{\text{pno}=2}(\text{Supply}) \]

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

Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

Clustered Index join

Unclustered index lookup Supply(pno)

T(Supplier) = 10000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
```plaintext
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PHYSICAL PLAN 3

\[ \text{Cost of Supply(pno)} = 4 \]
\[ \text{Cost of Index join} = 4 \]
\[ \text{Total cost:} 8 \]

Clustered Index join
\[ \sigma_{\text{scity}='Seattle' \land \text{sstate}='WA'} \]

\[ \sigma_{\text{pno}=2} \]

Unclustered index lookup
Supply(pno)

T(Supplier) = 10000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

\[ \text{T(Supply)} = 10000 \]
\[ \text{B(Supply)} = 100 \]
\[ \text{V(Supply, pno)} = 2500 \]```
Physical Plan 3

\[ \pi_{\text{sname}}(\sigma_{\text{pno}=2}(\text{Supplier})) = 1000 \]

\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]

\[ \text{Cost of Supply(pno)} = 4 \]
\[ \text{Cost of Index join} = 4 \]
\[ \text{Total cost: 8} \]

\[ \text{M=11} \]
QUERY OPTIMIZER

SUMMARY

Input: A logical query plan

Output: A good physical query plan

Basic query optimization algorithm

- Enumerate alternative plans (logical and physical)
- Compute estimated cost of each plan
- Choose plan with lowest cost

This is called cost-based optimization
DISK SCHEDULING

• Query optimization
  • Good DB design
  • Good estimation
  • Hardware independent

• All Disk I/Os are not created equal
  • Sectors close to each other are more preferable to read