CSE 344

JULY 20TH

COST ESTIMATION
ADMINISTRIVIA

• Midterm in one week
  • covers the material through today
    • Relational data model & queries
      • SQL, RA, Datalog
    • NoSQL data model and queries
    • Query optimization
  • more details next week...
WHICH INDEXES?

The *index selection problem*

- Given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)

Who does index selection:

- The database administrator DBA
- Semi-automatically, using a database administration tool
TWO TYPICAL KINDS OF QUERIES

- Point queries
  - (or equijoins)
  - Hash table or B+ tree

- Range queries
  - B+ tree only

SELECT * FROM Movie WHERE year = ?

SELECT * FROM Movie WHERE year >= ? AND year <= ?
BASIC INDEX SELECTION GUIDELINES

Consider queries in workload in order of importance

Consider relations accessed by query
  • No benefit to indexing other relations

Look at WHERE clause and JOIN .. ON for possible search key

Try to choose indexes that speed-up multiple queries
TO CLUSTER OR NOT

Range queries benefit mostly from clustering

Point indexes on keys do not need to be clustered
• will read 1 block whether the index is clustered or not

More generally, cost depends on percent of tuples returned...
SELECT * 
FROM R 
WHERE R.K>? and R.K<?
COST ESTIMATION

To estimate the cost of a query plan, we need to consider:

• How each operator is implemented
• The cost of each operator

Let’s start with the basics...
**COST PARAMETERS**

Cost = Disk I/O + Network I/O + Memory I/O + CPU

- Disk I/O $\gg$ Network I/O $\gg$ Memory I/O $\gg$ CPU
- We will focus on Disk I/O for now
  - if the query plan involves disk I/O, that is likely to dominate cost
  - for parallel, in-memory DBs, network costs usually dominate
COST PARAMETERS

Cost = Disk I/O + Network I/O + Memory I/O + CPU

• We will focus on Disk I/O for now

Parameters (a.k.a. statistics):

• $B(R) = \# \text{ of blocks for relation } R$
• $T(R) = \# \text{ of tuples in relation } R$
• $V(R, A) = \# \text{ of distinct values of attribute } A \text{ appearing in relation } R$

When $A$ is a key, $V(R,A) = T(R)$
When $A$ is not a key, $V(R,A)$ can be anything $\leq T(R)$
COST PARAMETERS

Cost = Disk I/O + Network I/O + Memory I/O + CPU

- We will focus on Disk I/O for now

Parameters (a.k.a. statistics):

- \( B(R) = \# \text{ of blocks for relation } R \)
- \( T(R) = \# \text{ of tuples in relation } R \)
- \( V(R, A) = \# \text{ of distinct values of attribute } A \text{ appearing in relation } R \)

DBMS collects statistics about base tables
must infer them for intermediate results

- (above information and more)
- allows DB to estimate things like “selectivity”...
SELECTIVITY FACTORS FOR CONDITIONS

A = c

\[ \sigma_{A=c}(R) \]

- Selectivity = \( \frac{1}{V(R,A)} \)

A < c

\[ \sigma_{A<c}(R) \]

- Selectivity = \( \frac{c - \min(R, A)}{\max(R, A) - \min(R, A)} \)

c1 < A < c2

\[ \sigma_{c1<A<c2}(R) \]

- Selectivity = \( \frac{c2 - c1}{\max(R, A) - \min(R, A)} \)
COST OF READING DATA FROM DISK

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

Index-based selection

- Estimate selectivity factor $f$ (see previous slide)
- Clustered index: $f \times B(R)$
- Unclustered index $f \times T(R)$

Note: we ignore I/O cost for index pages
INDEX BASED SELECTION

Example:

- \( B(R) = 2000 \)
- \( T(R) = 100,000 \)
- \( V(R, a) = 20 \)

Table scan:

Index based selection:

\[ \sigma_{a=v}(R) = ? \]
INDEX BASED SELECTION

Example:

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

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

Index based selection:

\[ \sigma_{a=v} \]

\[ R \]

Cost of \( \sigma_{a=v}(R) = ? \)
INDEX BASED SELECTION

Example:

- Table scan: $B(R) = 2,000$ I/Os
- $T(R) = 100,000$
- $V(R, a) = 20$

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

Index based selection:

- If index is clustered:
- If index is unclustered:
INDEX BASED SELECTION

Example:

- B(R) = 2000
- T(R) = 100,000
- V(R, a) = 20

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

Index based selection:

- If index is clustered: B(R) * 1/V(R,a) = 100 I/Os
- If index is unclustered:

\[
\sigma_{a=v} \quad \text{cost of } \sigma_{a=v}(R) = ?
\]
INDEX BASED SELECTION

Example:

\[ B(R) = 2000 \]
\[ T(R) = 100,000 \]
\[ V(R, a) = 20 \]

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

Index based selection:

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

\[ \sigma_{a=v} \]
\[ R \]

\text{cost of } \sigma_{a=v}(R) = ?
INDEX BASED SELECTION

Example:

- $B(R) = 2000$
- $T(R) = 100,000$
- $V(R, a) = 20$

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

Index based selection:

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

$\sigma_{a=v} \mid R$

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

Key Lesson: Don’t build unclustered indexes when $V(R,a)$ is small!
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 $S$ into hash table in main memory
- Then scan $R$ and join

Cost: $B(R) + B(S)$

Which relation to build the hash table on?
- (either can be done)
HASH JOIN EXAMPLE

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

1. Bob, Seattle
2. Ela, Everett
3. Jill, Kent
4. Joe, Seattle

2. Blue, 123
4. Prem, 432
3. Prem, 343
4. GrpH, 554

Two tuples per page
# HASH JOIN EXAMPLE

## Patient $\times$ Insurance

<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>

**Memory M = 21 pages**

**Showing pid only**

**Some large-enough #**

**This is one page with two tuples**
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

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

Input 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

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()`

<table>
<thead>
<tr>
<th>Disk</th>
<th>Patient</th>
<th>Insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</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>

<table>
<thead>
<tr>
<th>Memory M = 21 pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash h: pid % 5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>5 1 6 2 3 8 4 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input buffer</th>
<th>Output buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4</td>
<td>4 4</td>
</tr>
</tbody>
</table>
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

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

- multiple-pass since $S$ is read many times
- factor of $T(R)$ is very painful...
PAGE-AT-A-TIME REFINEMENT

```plaintext
for each page of tuples r in R do
  for each page 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: $B(R) + B(R)B(S)$

- only outer loops are disk I/O... inner-most loop is “free” (CPU)
- can speed this up even more if more memory is available
PAGE-AT-A-TIME REFINEMENT

Disk

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

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>
</tbody>
</table>

Input buffer for Patient

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

Input buffer for Insurance

Output buffer

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

<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
**PAGE-AT-A-TIME REFINEMENT**

<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:

- 1 2

Input buffer for Insurance:

- 2 8

Keep going until read all of Insurance

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

Output buffer:

- 2 2

Cost: \( B(R) + B(R)B(S) \)
SORT-MERGE JOIN

Sort-merge join:  $R \bowtie S$
- Scan $R$ and sort (in main memory if possible)
- Scan $S$ and sort (in main memory if possible)
- Merge $R$ and $S$ in one pass

Cost: $B(R) + B(S)$ if sorting done in memory
- only possible if $B(R) + B(S) \leq M$ (memory size)
- however, usually no more than 4x this when on disk
SORT-MERGE JOIN
EXAMPLE

Step 1: Scan Patient and sort in memory

Memory M = 21 pages

Disk

Patient    Insurance

1 2
3 4
9 6
8 5

2 4 6 6
4 3 1 3
2 8
8 9
SORT-MERGE JOIN
EXAMPLE

Step 2: Scan Insurance and sort in memory

Memory M = 21 pages

Disk

Patient Insurance

1 2
3 4
9 6
8 5

2 4
4 3
2 8
8 9

6 6
1 3
1 3
8 9

1 2
2 3
3 4
4 6

1 2
3 4
5 6
8 9

6 8
8 9
SORT-MERGE JOIN
EXAMPLE

Step 3: Merge Patient and Insurance

Disk

Patient

Insurance

Memory M = 21 pages

Output buffer
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

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

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</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>

Keep going until end of either 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:
  \[ B(R) + T(R) \times (T(S) \times \frac{1}{V(S,A)}) \]
- If A is a key, then both are \( B(R) + T(R) \)
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
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

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

LOGICAL QUERY PLAN 1

π_{sname}

σ_{pno=2 \land scity='Seattle' \land sstate='WA'}

T = 10000

sid = sid

Supplier

Supply
LOGICAL QUERY PLAN 1

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

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
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

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

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
   AND y.pno = 2
   AND x.scity = 'Seattle'
   AND x.sstate = 'WA'

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

**PHYSICAL PLAN 1**

\[
\begin{align*}
&\text{T > 1} \\
&\sigma_{pno=2 \land scity='Seattle' \land sstate='WA'} \\
&T = 10000 \\
&\text{sid = sid} \\
&\text{Block nested loop join} \\
&\text{Scan} \quad \text{Supply} \\
&\text{Scan} \quad \text{Supplier} \\
\end{align*}
\]

\[
\begin{align*}
&T(\text{Supply}) = 10000 \\
&B(\text{Supply}) = 100 \\
&V(\text{Supply, pno}) = 2500 \\
&T(\text{Supplier}) = 1000 \\
&B(\text{Supplier}) = 100 \\
&V(\text{Supplier, scity}) = 20 \\
&V(\text{Supplier, state}) = 10 \\
\end{align*}
\]

Total cost:

\[100 \times 10 = 1000\]
**PHYSICAL PLAN 1**

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

\[ \Pi_{sname} \]

\[ T < 1 \]

\[ T = 10000 \]

Block nested loop join

Supply

Scan

Cost: \( B(R) + B(R)B(S) \)

Total cost: \( 100 + 100 \times 100 = 10,100 \)

**Supplier(sid, sname, scity, sstate)**

**Supply(sid, pno, quantity)**

\[ T(Supply) = 10000 \]

\[ B(Supply) = 100 \]

\[ V(Supply, pno) = 2500 \]

\[ T(Supplier) = 1000 \]

\[ B(Supplier) = 100 \]

\[ V(Supplier, scity) = 20 \]

\[ V(Supplier, state) = 10 \]
**Physical Plan 2**

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

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

```
T = 4

T = 4

Main memory join

Unclustered index lookup
Supply(pno)

σ_{pno=2}

σ_{sid=sid}

σ_{sname}

T = 5

T = 5

Cost of Supply(pno) = 4

Cost of Supplier(scity) = 50

Total cost: 54

Cost of Supply(pno) = Cost of Supplier(scity) =
```

```
Unclustered index lookup
Supplier(scity)

σ_{sstate='WA'}

σ_{scity='Seattle'}

T = 50
```

```
T = 4
```

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

```
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
```
\[ \text{Supply}(\text{sid, pno, quantity}) \]
\[ \text{Supplier}(\text{sid, sname, scity, sstate}) \]

**PHYSICAL PLAN 2**

\[ \text{σ}_{\text{sstate} = 'WA'}(\text{Supplier}) \]
\[ \text{T}(\text{Supplier}) = 1000 \]
\[ \text{B}(\text{Supplier}) = 100 \]
\[ \text{V}(\text{Supplier, scity}) = 20 \]
\[ \text{V}(\text{Supplier, state}) = 10 \]

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

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

### Physical Plan 2

- **Unclustered index lookup**
  - $\sigma_{\text{pno}=2}$
  - $\text{Supply}$
  - $T=4$

- **Main memory join**
- **Unclustered index lookup**
  - $\sigma_{\text{sstate}='WA'}$
  - $\text{Supplier}$
  - $T=5$

- **Unclustered index lookup**
  - $\sigma_{\text{scity}='Seattle'}$
  - $\text{Supplier}$

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

- $T(\text{Supply}) = 10000$
- $B(\text{Supply}) = 100$
- $V(\text{Supplier}, \text{pno}) = 2500$
- $T(\text{Supplier}) = 1000$
- $B(\text{Supplier}) = 100$
- $V(\text{Supplier}, \text{scity}) = 20$
- $V(\text{Supplier}, \text{sstate}) = 10$
Supplier(sid, name, city, state)
Supply(sid, pno, quantity)

**PHYSICAL PLAN 3**

\[
\text{π}_{\text{name}}(\text{Supplier}) = 1000
\]

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

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

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

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

\[
\text{Cost of Index join} = 4
\]

\[
\text{Total cost} = 8
\]

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

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

\[
\text{V(Supply, pno)} = 2500
\]
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

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

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

**PHYSICAL PLAN 3**

\[ \pi_{sname}(\sigma_{pno=2}(\text{Supply})) = 1000 \]
\[ \text{B(Supplier)} = 100 \]
\[ \text{V(Supplier, scity)} = 20 \]
\[ \text{V(Supplier, state)} = 10 \]

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

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