CS 245: Database System Principles

Notes 7: Query Optimization
Chris Olston
Hector Garcia-Molina

System Structure

queries

- Query Optimization and Execution
- Relational Operators
- Files and Access Methods
- Buffer Management
- Disk Space Management

Conciseness Control & Recovery

DB

Metamorphosis: From Query to Plan

1:n

- Declarative query
  - SQL
- Logical query plan
  - Relational algebra
- Physical query plan
  - What actually gets executed

SELECT
FROM
WHERE

hash-join
scan
scan

1:n

Query Interface

History

- In the "dark days," the application programmer would program the execution plan
- Problems
  - More work for application programmer
  - Have to re-program to handle changes to physical design (e.g., remove an index)
  - Optimal execution plan may change over time

Declarative Query Languages

Now

- Relational databases have query language interface
- Declarative specification:
  - Say what you want, not how to get it

Rely on the query optimizer to pick the best plan...

Query Optimization Strategies

1) Guess
   - Could be orders of magnitude slower than the optimal plan (hours instead of seconds)
2) Ask user’s advice
   - User does not know about physical design of DB
3) Execute all options, measure, and pick fastest
   - Optimal, but obviously not the way to go
4) Be smart!
Relational Query Optimization

- Enumerate plans;
- Estimate costs;
- Pick cheapest

- Relies on cost estimation
- Costs estimated using:
  - Statistics about relations in DB
  - Input size estimates

Motivation: Input size estimation

- Cost of an operation (join, sort, ...) depends on sizes of inputs

\[
\text{Need: size}(R), \text{size}(S) \leq |T|
\]

\[
\text{Need: size}(S), \text{size}(T)
\]

Estimating input size

- Keep statistics for relation R
  - \( T(R) = |R| \) = # tuples in R
  - \( S(R) \) = # of bytes in each R tuple
  - \( B(R) \) = # of blocks to hold all R tuples
  - \( V(R, A) \) = # distinct values in R for attribute A

Example

\[
\begin{array}{c|c|c|c|c}
R & A & B & C & D \\
\hline
\text{cat} & 1 & 10 & a & \\
\text{cat} & 1 & 20 & b & \\
\text{dog} & 1 & 30 & a & \\
\text{dog} & 1 & 40 & c & \\
\text{bat} & 1 & 50 & d & \\
\end{array}
\]

- A: 20 byte string
- B: 4 byte integer
- C: 8 byte date
- D: 5 byte string

\[
T(R) = 5 \quad S(R) = 37
\]

\[
V(R,A) = 3 \quad V(R,C) = 5
\]

\[
V(R,B) = 1 \quad V(R,D) = 4
\]

Size estimates for \( W = R1 \times R2 \)

\[
T(W) = T(R1) \times T(R2)
\]

\[
S(W) = S(R1) + S(R2)
\]
**Size estimate** for \( W = \sigma_{\text{val}}(R) \)

\[ S(W) = S(R) \]

\[ T(W) = ? \]

---

**Example**

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>1</td>
<td>30</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>1</td>
<td>40</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

\[ V(R,A) = 3 \]

\[ V(R,B) = 1 \]

\[ V(R,C) = 5 \]

\[ V(R,D) = 4 \]

\[ W = \sigma_{\text{val}}(R) \quad T(W) = \frac{T(R)}{V(R,Z)} \]

---

**Assumption:**

Values in select expression \( Z = \text{val} \) are **uniformly distributed** over possible \( V(R,Z) \) values.

---

**Alternate Assumption:**

Values in select expression \( Z = \text{val} \) are **uniformly distributed** over domain with \( \text{DOM}(R,Z) \) values.

---

**Example**

Alternate assumption

\[ V(R,A) = 3 \quad \text{DOM}(R,A) = 10 \]

\[ V(R,B) = 1 \quad \text{DOM}(R,B) = 10 \]

\[ V(R,C) = 5 \quad \text{DOM}(R,C) = 10 \]

\[ V(R,D) = 4 \quad \text{DOM}(R,D) = 10 \]

\[ W = \sigma_{\text{val}}(R) \quad T(W) = ? \]

---

\[ C = \text{val} \Rightarrow T(W) = (1/10)1 + (1/10)1 + ... = (5/10) = 0.5 \]

\[ B = \text{val} \Rightarrow T(W) = (1/10)5 + 0 + 0 = 0.5 \]

\[ A = \text{val} \Rightarrow T(W) = (1/10)2 + (1/10)2 + (1/10)1 = 0.5 \]
Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alternate assumption:

\[ V(R,A) = 3 \quad \text{DOM}(R,A) = 10 \]
\[ V(R,B) = 1 \quad \text{DOM}(R,B) = 10 \]
\[ V(R,C) = 5 \quad \text{DOM}(R,C) = 10 \]
\[ V(R,D) = 4 \quad \text{DOM}(R,D) = 10 \]

\[ W = \sigma_{z = \text{val}(R)} \quad T(W) = \frac{T(R)}{\text{DOM}(R, Z)} \]

Selection cardinality

\[ SC(R,A) = \text{average \# records that satisfy equality condition on R,A} \]
\[ SC(R,A) = \begin{cases} T(R) & \text{if } V(R,A) \leq W \\ \cdot & \text{otherwise} \end{cases} \]

\[ \text{DOM}(R,A) \]

What about \( W = \sigma_{z \geq \text{val}(R)} \)?

- Solution # 1:
  \[ T(W) = \frac{T(R)}{2} \]
- Solution # 2:
  \[ T(W) = \frac{T(R)}{3} \]

\[ \sigma_{z \geq \text{val}(R)} \]

Solution # 3: Estimate values in range

Example

\[ R \]
\[ \begin{array}{|c|c|} \hline \text{Z} & \text{Min} = 1 \\ \text{Max} = 20 \hline \end{array} \]
\[ W = \sigma_{z \geq \text{val}(R)} \]

\[ f = 20 - 15 + 1 = 6 \quad \frac{15-10}{20-1+1} = \frac{20}{20} \]

\[ T(W) = f \times T(R) \]

Size estimate for \( W = R_1 \bowtie R_2 \)

Let \( x \) = attributes of \( R_1 \)
Let \( y \) = attributes of \( R_2 \)

Case 1

\[ X \cap Y = \emptyset \]

Same as \( R_1 \times R_2 \)

Case 2

Assumption:

\[ V(R_1, A) \leq V(R_2, A) \Rightarrow \text{Every A value in } R_1 \text{ is in } R_2 \]
\[ V(R_2, A) \leq V(R_1, A) \Rightarrow \text{Every A value in } R_2 \text{ is in } R_1 \]

"Containment of value sets" Sec. 7.4.4
**Computing** $T(W)$ **when** $V(R1,A) \leq V(R2,A)$

<table>
<thead>
<tr>
<th>R1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>R2</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Take 1 tuple

1 tuple matches with $\frac{T(R2)}{V(R2,A)}$ tuples...

so

$T(W) = \frac{T(R2) \times T(R1)}{V(R2,A)}$

- $V(R1,A) \leq V(R2,A)$
  - $T(W) = \frac{T(R2) \times T(R1)}{V(R2,A)}$
- $V(R2,A) \leq V(R1,A)$
  - $T(W) = \frac{T(R1) \times T(R2)}{V(R1,A)}$

[A is common attribute]

---

**In general** $W = R1 \bowtie R2$

$T(W) = \frac{T(R1) \times T(R2)}{\max\{V(R1,A), V(R2,A)\}}$

---

**Case 2** with alternate assumption

Values uniformly distributed over domain

<table>
<thead>
<tr>
<th>R1</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>R2</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This tuple matches $T(R2)/\text{DOM}(R2,A)$ so

$T(W) = \frac{T(R2) \times T(R1)}{\text{DOM}(R2,A)} \times \frac{T(R2) \times T(R1)}{\text{DOM}(R1,A)}$

---

**In all cases:**

$S(W) = S(R1) + S(R2) - S(A)$

size of attribute A

---

**Using similar ideas,**
we can estimate sizes of:

- $\Pi_{=\alpha} (R)$ ..... Sec. 7.4.2
- $\sigma_{A=\alpha} (R)$ ..... Sec. 7.4.3
- $R \bowtie S$ with common attrs. A,B,C
  - Sec. 7.4.5
- Union, intersection, diff, .... Sec. 7.4.7
Note: for complex expressions, need intermediate T,S,V results.

E.g. \( W = [\sigma_{A=n}(R1) \bowtie R2 \] T(U) = T(R1)/V(R1,A) \quad S(U) = S(R1)

Also need \( V(U, *) \) !!

To estimate \( V_s \)

E.g., \( U = \sigma_{A=a}(R1) \)

Say R1 has attrs A,B,C,D

\[ V(U, A) = \]

\[ V(U, B) = \]

\[ V(U, C) = \]

\[ V(U, D) = \]

Possible Guess \( U = \sigma_{A=a}(R) \)

\[ V(U, A) = 1 \]

\[ V(U, B) = V(R, B) \]

Example:

\[ U = R1(A,B) \bowtie R2(A,C) \]

\[ V(U, A) = \min \{ V(R1, A), V(R2, A) \} \]

\[ V(U, B) = V(R1, B) \]

\[ V(U, C) = V(R2, C) \]

[called "preservation of value sets" in section 7.4.4]

Example:

\[ Z = R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D) \]

\[ R1 \]

\[ T(R1) = 1000 \quad V(R1,A)=50 \quad V(R1,B)=100 \]

\[ R2 \]

\[ T(R2) = 2000 \quad V(R2,B)=200 \quad V(R2,C)=300 \]

\[ R3 \]

\[ T(R3) = 3000 \quad V(R3,C)=90 \quad V(R3,D)=500 \]
Partial Result: \( U = R \bowtie S \)

\[
\begin{align*}
T(U) &= \frac{1000 \times 2000}{200} \\
V(U,A) &= 50 \\
V(U,B) &= 100 \\
V(U,C) &= 300
\end{align*}
\]

\[
\begin{align*}
Z = U \bowtie R_3
\end{align*}
\]

\[
\begin{align*}
T(Z) &= \frac{1000 \times 2000 \times 3000}{200 \times 300} \\
V(Z,A) &= 50 \\
V(Z,B) &= 100 \\
V(Z,C) &= 90 \\
V(Z,D) &= 500
\end{align*}
\]

**Summary**

- Estimating size of results is an “art”
- Don’t forget:
  - Statistics must be kept up to date...
    - (cost?)

**Outline**

- Estimating cost of query plan
  - Estimating size of inputs — done!
  - Cost estimation next

**Cost Estimation**

Cost depends on input size/distribution and other parameters such as:

- \( M \) = # memory blocks available
- \( HT(i) \) = # levels in index \( i \)
- \( LB(i) \) = # of leaf blocks in index \( i \)

**Cost Formulas**

- From Notes 6 ...
  - Nested-Loops Join \((R \bowtie S)\):  
    \[
    \text{Cost} = B(R) + \left\lceil \frac{B(R)}{M-2} \right\rceil B(S)
    \]
- **Index-NL (R ∩ S)**, index on S.A):
  
  Cost = B(R) + |R|·HT(S-Index) + \(\frac{|R|·|S|}{\max(V(R,A), V(S,A))}\)

  (unless part of index buffer)

  (unless S is clustered)

- **Sort-Merge Join (R ∩ S)**:
  
  Cost to sort R into runs = 2·B(R)
  Cost to sort S into runs = 2·B(S)
  Cost to merge R & S runs = B(R) + B(S)

  Overall Cost = 3·(B(R) + B(S))

- **Hash Join (R ∩ S)**:
  
  Cost = 3·(B(R) + B(S))

  (unless keep some buckets in memory:
   hybrid hash join)

**Outline**

- Estimating cost of query plan — done!

- Generating and comparing plans — next

**Generate & Compare Plans: One Idea**

1. Enumerate all possible plans
2. Estimate costs
3. Pick best one

   Problem: too many plans

   Observation: some plans are almost always bad...

**Generate & Compare Plans: Better Idea**

1. Selectively enumerate logical plans (using heuristics) — next
2. Enumerate physical alternatives
3. Estimate costs
4. Pick best one
Relational algebra transformations

- Transformation rules (preserve equivalence)
- What are good transformations?

Rules: Natural joins & cross products & union

\[ R \bowtie S = S \bowtie R \]
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]

Note:

- Carry attribute names in results, so order is not important
- Can also write as trees, e.g.:

\[
\begin{array}{ccc}
\bowtie & \bowtie & = \\
\bowtie & \bowtie & \bowtie \\
R & S & T \\
\end{array}
\]

Rules: Selects

\[ \sigma_{p1,p2}(R) = \sigma_{p1} [ \sigma_{p2} (R)] \]
\[ \sigma_{p1p2}(R) = \text{See textbook} \]
\[(\text{gets complicated with bag semantics})\]

Rules: Project

Let: \( X = \) set of attributes
\( Y = \) set of attributes
\( XY = X \cup Y \)
\[ \pi_{xy} (R) = \pi_{x} [\pi_{y} (R)] \]
**Rules: \( \sigma + \bowtie \) combined**

Let \( p \) = predicate with only \( R \) attribs

\( q \) = predicate with only \( S \) attribs

\( m \) = predicate with only \( R, S \) attribs

\[
\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S
\]

\[
\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]
\]

**Rules: \( \sigma + \bowtie \) combined** (continued)

Some Rules can be Derived:

\[
\sigma_{p \land q} (R \bowtie S) =
\]

\[
\sigma_{p \land q} (R \bowtie S) =
\]

\[
\sigma_{p \lor q} (R \bowtie S) =
\]

---

**Do one, others left as exercises:**

\[
\sigma_{p \land q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)]
\]

\[
\sigma_{p \land q \land m} (R \bowtie S) =
\]

\[
\sigma_{m} [(\sigma_p R) \bowtie (\sigma_q S)]
\]

\[
\sigma_{p \lor q} (R \bowtie S) =
\]

\[
[(\sigma_p R) \bowtie S] \cup [R \bowtie (\sigma_q S)]
\]

**--\(\) Derivation for first one:**

\[
\sigma_{p \land q} (R \bowtie S) =
\]

\[
\sigma_p [\sigma_q (R \bowtie S)] =
\]

\[
\sigma_p [R \bowtie \sigma_q (S)] =
\]

\[
[\sigma_p (R)] \bowtie [\sigma_q (S)]
\]

**Rules: \( \pi, \sigma \) combined**

Let \( x \) = subset of \( R \) attributes

\( z \) = attributes in predicate \( P \)

(subset of \( R \) attributes)

\[
\pi_x [\sigma_p (R)] = \pi_x \{\sigma_p \{\pi_x (R)\}\}
\]

**Rules: \( \pi, \bowtie \) combined**

Let \( x \) = subset of \( R \) attributes

\( y \) = subset of \( S \) attributes

\( z \) = intersection of \( R, S \) attributes

\[
\pi_{xy} (R \bowtie S) =
\]

\[
\pi_{xy} \{[\pi_x (R)] \bowtie [\pi_y (S)]\}
\]
\[ \pi_{xy} \{ \sigma_p (R \bowtie S) \} = \]
\[ \pi_{xy} \{ \sigma_p [\pi_{x'} (R) \bowtie \pi_{y'} (S)] \} \]
\[ z' = z \cup \{ \text{attributes used in } P \} \]

**Rules** for \( \sigma, \pi \) combined with \( X \)

similar...

e.g., \( \sigma_p (R \times S) = ? \)

---

**Rules** \( \sigma, U \) combined:

- \( \sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S) \)
- \( \sigma_p(R - S) = \sigma_p(R) - S = \sigma_p(R) - \sigma_p(S) \)

---

**In textbook:** more transformations

- Eliminate common sub-expressions
- Other operations: duplicate elimination

---

Which are "good" transformations?

- \( \sigma_{p1 \cdot p2} (R) \rightarrow \sigma_{p1} [\sigma_{p2} (R)] \)
- \( \sigma_p (R \bowtie S) \rightarrow [\sigma_p (R)] \bowtie S \)
- \( R \bowtie S \rightarrow S \bowtie R \)
- \( \pi \{ \sigma_p (R) \} \rightarrow \pi \{ \sigma_p [\pi_{x'} (R)] \} \)

---

**Bottom line:**

- There are some transformations that are usually good
- BUT: No transformation is always good

- Tradeoff:
  
  reduced search space \( \iff \) loss of optimality
System R Optimizer Heuristics (usually good)
- Push down selections & projections
- Postpone cross-products
- Only consider "left-deep" query plans

System R Optimizer: Transformations
- Left/right join inputs
  - RS
  - SR
- Join order
  - RT
  - TR
  - TS
  - ST
- Physical alternatives
  - Access methods (seq. scan, index lookup, ...)
  - Join strategies (nested-loops, hash join, ...)

Query "Good-izer"
- Use of heuristics → can’t claim optimality
- Instead, hope to get a fairly good plan
- Goal: eliminate horrible plans that take hours instead of seconds

Generate & Compare Plans
1. Selectively enumerate logical plans (using heuristics)
2. Enumerate physical alternatives
3. Estimate costs
4. Pick best one

Generate & Compare Plans: Improved
Combine enumeration & cost estimation
1. Enumerate small sub-plans & estimate costs
2. Prune (remove) "sub-optimal" alternatives
3. Enumerate ways to assemble sub-plans into larger sub-plans & estimate costs
4. Prune again (keep only "optimal" sub-plans)
   ... Keep building larger "optimal" sub-plans
   ... Eventually generate "optimal" overall plan

Dynamic Programming Approach
- Level 1 sub-plan: join of 2 relations (plus access methods)
  - Index-NL
  - SeqScan(R)
  - IndexLookup(S)
- Level 2 sub-plan: join of 3 relations
  - Hash
  - Index-NL
  - Hash
  - Sort-Merge
- Level n sub-plan: join of n+1 relations
Generating Logical Sub-Plans

- Only generate logical sub-plans that conform to heuristic rules

Example:

\[
\begin{array}{c}
\text{R} & \text{S} \\
\hline
\text{R.a} = \text{S.b} & \text{q(S.b=5)} \\
\text{q(R.a = R.b)} & \text{q(S.b=5)} \\
\end{array}
\]

Generating Physical Sub-Plans

- For each heuristically chosen logical sub-plan, try all combinations of physical alternatives

Example:

Example (cont.)

- Memory size: \( M = 102 \)
- Index on R.A; all non-leaves fit in memory
- S.A foreign key onto R.A

For simplicity:

- Assume: join tuple size is sum of sizes of component tuples
- Assume: always write out intermediate results
- Consider the following join strategies:
  - Nested-loops
  - Index nested-loops
  - Sort-merge join

Example:

\[
\text{SELECT *}
\]
\[
\text{FROM R, S, T}
\]
\[
\text{WHERE R.A = S.A and S.B = T.B}
\]

- \(|R| = 30,000; B(R) = 300\)
- \(|S| = 100,000; B(S) = 1000\)
- \(|T| = 20,000; B(T) = 200\)
- \(V(S, B) = 25,000; V(T, B) = 10,000\)
**Level-1 Sub-Plans (no X-prods)**

- Merge-Join
- Sort(R)
- Sort(S)
- SeqScan(R)
- SeqScan(S)
- Index-IL
- SeqScan(S)

**Costs**

- Merge-Join
- Sort(R) Sort(S)
- Nested-Loops
- SeqScan(R) SeqScan(S)
- SeqScan(T)
- Index-IL
- SeqScan(S)

**Cost Formulas**

- Cost = \(3 \cdot (B(R) + B(T)) = 3600\)
- Cost = \(B(R) + \frac{B(S)}{M-2} \cdot B(T) = 3000\)
- Cost = \(B(T) + \frac{B(S)}{M-2} \cdot B(R) = 2200\)

---

**Pruning Level-1 Sub-Plans**

- Merge-Join
- Sort(R) Sort(S) Sort(T)
- Nested-Loops
- SeqScan(R) SeqScan(S) SeqScan(T)
- SeqScan(S) SeqScan(R)
- Index-IL

**Costs**

- Merge-Join
- Sort(R) Sort(S) Sort(T)
- Nested-Loops
- SeqScan(R) SeqScan(S) SeqScan(T)
- SeqScan(S) SeqScan(R)
- Index-IL

**Cost Formulas**

- Cost = \(3 \cdot (B(R) + B(S)) - 3900\)
- Cost = \(B(R) + \frac{B(S)}{M-2} \cdot B(T) = 3300\)
- Cost = \(B(S) + \frac{B(R)}{M-2} \cdot B(T) = 4000\)
- Cost = \(B(S) + |S| + |T| = 201,000\)

---

**What is size of R \(\bowtie\) S?**

Recall: S.A foreign key onto R.A

- \(|R \bowtie S| = |S|\)
- \(B(RS) = 2 \cdot B(S)\)

---

**Level-2 Sub-Plans (left-deep)**

- Merge-Join
- Sort
- Nested-Loops
- SeqScan(R) SeqScan(S) SeqScan(T)

**Costs**

- Merge-Join
- Sort
- Nested-Loops
- SeqScan(R) SeqScan(S) SeqScan(T)

**Cost Formulas**

- Cost = \(3300 + B(RS) + 3 \cdot (B(RS) + B(T)) = 11,900\)
- Cost = \(3300 + B(RS) + \frac{B(RS)}{M-2} \cdot B(T) = 11,300\)
**What is size of \( S \times T \)?**

\[
|S \times T| = \frac{|S| \cdot |T|}{\max(V(S,B), V(T,B))} = 80,000
\]

\[B(ST) = 80,000/50 = 1600\]

**Level-2 Sub-Plans (cont.)**

\[
B(ST) = 1600
\]

**Winner**

\[
\begin{array}{c}
\text{Nested-Loops} \\
\text{SeqScan(T)} \\
\text{SeqScan(S)} \\
\text{SeqScan(R)}
\end{array}
\]

- Selected plan: \( C = 10,200 \)
- Worse plan: \( C > 201,000 \)
- Benefit from optimizer: 20x speedup!

**Summary**

- Query Optimization
  - Statistics
  - Intermediate result size estimation
  - Cost estimation
  - Logical transformations & heuristics
  - Generating and comparing plans

**Readings in Textbook**

- Chapter 7, except:
  - 7.6, 7.7.2, 7.7.3, 7.7.4, 7.7.5, 7.7.6
  - Material on duplicate elimination operator, grouping, aggregation operators
- Note: The dynamic programming approach to enumeration and pruning is not included in reading list, but...
- You are responsible for understanding that material as covered in class!!!