Announcements

• Homework 5 is due next week
  – How is it going?

• Homework 6 (last) to be posted soon
  – Rather short assignment, but start early in case you have questions

• Final will be take-home
  – Posted on June 2\textsuperscript{nd}, after last lecture
  – Due by June 4\textsuperscript{th}; electronic turn-in
Where We Are

• We are learning how a DBMS executes a query
• What we learned so far
  – How data is stored and indexed: lecture 6
  – Logical query plans and physical operators: lecture 7
• Today
  – How to select logical & physical query plans

Note: Today’s material contains more than Chapter 15 in the textbook!
Query Optimization Goal

• For a query
  – There exists many logical and physical query plans
  – Query optimizer needs to pick a good one
Query Optimization Algorithm

• Enumerate alternative plans

• Compute estimated cost of each plan
  – Compute number of I/Os
  – Compute CPU cost

• Choose plan with lowest cost
  – This is called cost-based optimization
Example

Suppliers(sid, sname, scity, sstate)
Supplies(sid, pno, quantity)

• Some statistics
  – T(Supplier) = 1000 records
  – B(Supplier) = 100 pages
  – T(Supplies) = 10,000 records
  – B(Supplies) = 100 pages
  – V(Supplier,scity) = 20, V(Supplier,state) = 10
  – V(Supplies,pno) = 2,500
  – Both relations are clustered

• M = 10
Physical Query Plan 1

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

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

(Block-nested loop)
\[ \text{sid = sid} \]

Total cost of plan is thus cost of join:
\[ = B(\text{Supplier}) + B(\text{Supplier}) \times B(\text{Supplies})/M \]
\[ = 100 + 10 \times 100 \]
\[ = 1,100 \text{ I/Os} \]
Physical Query Plan 2

\[ \pi_{\text{sname}} \]

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

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

3. Join \( \text{sid} = \text{sid} \)

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

Total cost:

\[ = 100 + 100 \times \frac{1}{20} \times \frac{1}{10} \]
\[ + 100 + 100 \times \frac{1}{2500} \]
\[ + 2 \]
\[ + 0 \]

\[ \approx 204 \text{ I/Os} \]
Physical Query Plan 3

(On the fly)  (4) $\pi_{\text{sname}}$

(On the fly)

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

(2) $\text{sid} = \text{sid}$ (Index nested loop)

(Use index)

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

---

Supplies

(Index lookup on pno)

Assume: clustered

Suppliers

(Index lookup on sid)

 Doesn’t matter if clustered or not

Total cost

= 1 (1)
+ 4 (2)
+ 0 (3)
+ 0 (3)

Total cost $\approx 5$ I/Os

4 tuples
Simplifications

• In the previous examples, we assumed that all index pages were in memory

• When this is not the case, we need to add the cost of fetching index pages from disk
Lessons

• Need to consider several physical plan
  – even for one, simple logical plan
• No magic “best” plan: depends on the data
• In order to make the right choice
  – need to have statistics over the data
  – the B’s, the T’s, the V’s
Outline

• Search space

• Algorithm for enumerating query plans

• Estimating the cost of a query plan
Relational Algebra Equivalences

• Selections
  – Commutative: $\sigma_{c_1}(\sigma_{c_2}(R))$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$
  – Cascading: $\sigma_{c_1 \land c_2}(R)$ same as $\sigma_{c_2}(\sigma_{c_1}(R))$

• Projections

• Joins
  – Commutative: $R \bowtie S$ same as $S \bowtie R$
  – Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$
Left-Deep Plans and Bushy Plans

Left-deep plan

Bushy plan
Example:
Simple Algebraic Laws

• Commutative and Associative Laws
  \[ R \cup S = S \cup R, \quad R \cup (S \cup T) = (R \cup S) \cup T \]
  \[ R \Join S = S \Join R, \quad R \Join (S \Join T) = (R \Join S) \Join T \]
  \[ R \Join S = S \Join R, \quad R \Join (S \Join T) = (R \Join S) \Join T \]

• Distributive Laws
  \[ R \Join (S \cup T) = (R \Join S) \cup (R \Join T) \]
Example:
Simple Algebraic Laws

• Laws involving selection:

\[ \sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R) \]

\[ \sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R) \]

\[ \sigma_C (R \Join S) = \sigma_C (R) \Join S \]

• When C involves only attributes of R

\[ \sigma_C (R - S) = \sigma_C (R) - S \]

\[ \sigma_C (R \cup S) = \sigma_C (R) \cup \sigma_C (S) \]

\[ \sigma_C (R \Join S) = \sigma_C (R) \Join S \]
Example:  
Simple Algebraic Laws

• Example:  \( R(A, B, C, D), S(E, F, G) \)

\[
\sigma_{F=3} (R \bowtie_{D=E} S) = ? \\
\sigma_{A=5 \text{ AND } G=9} (R \bowtie_{D=E} S) = ?
\]
Example: Simple Algebraic Laws

• Laws involving projections
\[ \Pi_M(R \Join S) = \Pi_M(\Pi_P(R) \Join \Pi_Q(S)) \]
\[ \Pi_M(\Pi_N(R)) = \Pi_{M,N}(R) \]

• Example \( R(A,B,C,D), S(E, F, G) \)
\[ \Pi_{A,B,G}(R \Join_{D=E} S) = \Pi_? (\Pi_?(R) \Join_{D=E} \Pi_?(S)) \]
Example:
Simple Algebraic Laws

- Laws involving grouping and aggregation:
  \[ \delta(\gamma_{A, \text{agg}(B)}(R)) = \gamma_{A, \text{agg}(B)}(R) \]
  \[ \gamma_{A, \text{agg}(B)}(\delta(R)) = \gamma_{A, \text{agg}(B)}(R) \text{ if agg is “duplicate insensitive”} \]

- Which of the following are “duplicate insensitive”? sum, count, avg, min, max

\[ \gamma_{A, \text{agg}(D)}(R(A,B) \Join_{B=C} S(C,D)) = \gamma_{A, \text{agg}(D)}(R(A,B) \Join_{B=C} (\gamma_{C, \text{agg}(D)} S(C,D))) \]
Laws Involving Constraints

Product\((pid, pname, price, cid)\)
Company\((cid, cname, city, state)\)

\[ \Pi_{pid, price}(Product \bowtie_{cid=cid} Company) = \Pi_{pid, price}(Product) \]

Need a second constraint for this law to hold. Which one?
Laws with Semijoins

Recall the definition of a semijoin:

- \( R \bowtie S = \Pi_{A_1, \ldots, A_n} (R \bowtie S) \)

- Where the schemas are:
  - Input: \( R(A_1, \ldots, A_n), \ S(B_1, \ldots, B_m) \)
  - Output: \( T(A_1, \ldots, A_n) \)
Laws with Semijoins

Semijoins: a bit of theory (see *Database Theory, AHV*)

• Given a query:  
  \[ Q \leftarrow \Pi (\sigma (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)) \]

• A **semijoin reducer** for Q is

  \[
  \begin{align*}
  R_{i1} &:= R_{i1} \bowtie R_{j1} \\
  R_{i2} &:= R_{i2} \bowtie R_{j2} \\
  \ldots \ldots \\
  R_{ip} &:= R_{ip} \bowtie R_{jp}
  \end{align*}
  \]

  such that the query is equivalent to:

  \[ Q \leftarrow \Pi (\sigma (R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn})) \]

• A **full reducer** is such that no dangling tuples remain
Laws with Semijoins

• Example:

\[
Q(A,E) \leftarrow \Pi_{A,E}(R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D,E))
\]

• A full reducer is:

\[
\begin{align*}
R_2'(B,C) & := R_2(B,C) \bowtie R_1(A,B) \\
R_3'(C,D,E) & := R_3(C,D,E) \bowtie R_2(B,C) \\
R_2''(B,C) & := R_2'(B,C) \bowtie R_3'(C,D,E) \\
R_1'(A,B) & := R_1(A,B) \bowtie R_2''(B,C)
\end{align*}
\]

\[
Q(A,E) \leftarrow \Pi_{A,E}(R_1'(A,B) \bowtie R_2''(B,C) \bowtie R_3'(C,D,E))
\]

The new tables have only the tuples necessary to compute \(Q(E)\)
Laws with Semijoins

• Example:

\[ Q(E) ::= R1(A,B) \Join R2(B,C) \Join R3(A,C, E) \]

• Doesn’t have a full reducer (we can reduce forever)

**Theorem** a query has a full reducer iff it is “acyclic”

[Database Theory, by Abiteboul, Hull, Vianu]
Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

View:
CREATE VIEW DepAvgSal As (
    SELECT E.did, Avg(E.Sal) AS avgsal
    FROM Emp E
    GROUP BY E.did)

Query:
SELECT E.eid, E.sal
FROM Emp E, Dept D, DepAvgSal V
WHERE E.did = D.did AND E.did = V.did
    AND E.age < 30 AND D.budget > 100k
    AND E.sal > V.avgsal

Goal: compute only the necessary part of the view
Example with Semijoins

**Emp(eid, ename, sal, did)**
**Dept(did, dname, budget)**
**DeptAvgSal(did, avgsal) /* view */**

New view uses a reducer:

```
CREATE VIEW LimitedAvgSal As
    SELECT E.did, Avg(E.Sal) AS avgsal
    FROM Emp E, Dept D
    WHERE E.did = D.did AND D.budget > 100k
    GROUP BY E.did)
```

New query:

```
SELECT E.eid, E.sal
FROM Emp E, Dept D, LimitedAvgSal V
WHERE E.did = D.did AND E.did = V.did
    AND E.age < 30 AND D.budget > 100k
    AND E.sal > V.avgsal
```
Example with Semijoins

**CREATE VIEW** PartialResult **AS**

```
(SELECT E.eid, E.sal, E.did
FROM Emp E, Dept D
WHERE E.did = D.did AND E.age < 30
AND D.budget > 100k)
```

**CREATE VIEW** Filter **AS**

```
(SELECT DISTINCT P.did FROM PartialResult P)
```

**CREATE VIEW** LimitedAvgSal **AS**

```
(SELECT E.did, Avg(E.Sal) AS avgsal
FROM Emp E, Filter F
WHERE E.did = F.did GROUP BY E.did)
```

[Chaudhuri’98]

Full reducer:

**CREATE VIEW** PartialResult **AS**

```
(SELECT E.eid, E.sal, E.did
FROM Emp E, Dept D
WHERE E.did = D.did AND E.age < 30
AND D.budget > 100k)
```

**CREATE VIEW** Filter **AS**

```
(SELECT DISTINCT P.did FROM PartialResult P)
```

**CREATE VIEW** LimitedAvgSal **AS**

```
(SELECT E.did, Avg(E.Sal) AS avgsal
FROM Emp E, Filter F
WHERE E.did = F.did GROUP BY E.did)
```
Example with Semijoins

New query:

```
SELECT P.eid, P.sal
FROM PartialResult P, LimitedDepAvgSal V
WHERE P.did = V.did AND P.sal > V.avgsal
```
Search Space Challenges

• Search space is huge!
  – Many possible equivalent trees
  – Many implementations for each operator
  – Many access paths for each relation
    • File scan or index + matching selection condition

• Cannot consider ALL plans
  – Heuristics: only partial plans with “low” cost
Outline

• Search space

• Algorithms for enumerating query plans

• Estimating the cost of a query plan
Key Decisions

- When selecting a plan, some of the most important decisions include:
  - Logical plan
    - Which algebraic laws do we apply, and in which context(s)?
    - What logical plans do we consider (left-deep, bushy?)
  - Physical plan
    - What join algorithms to use?
    - What access paths to use (file scan or index)?
Optimizers

• Heuristic-based optimizers:
  – Apply greedily rules that always improve
    • Typically: push selections down
  – Very limited: no longer used today

• Cost-based optimizers
  – Use a cost model to estimate the cost of each plan
  – Select the “cheapest” plan
Representation of Partial Plans

• Bottom-up optimization algorithms:
  – A partial plan is an algebra tree that computes only part of the query

• Top-down optimization algorithms:
  – A partial plan is an algebra tree whose leaves are either base relations, or queries (without a plan yet)
Examples of Partial Plans

\[
\begin{align*}
\text{SELECT} & \quad * \\
\text{FROM} & \quad \text{R, S, T} \\
\text{WHERE} & \quad \text{R.B=S.B and S.C=T.C and R.A<40}
\end{align*}
\]

Bottom-up plans

\[
\begin{align*}
\sigma_{A<40} \quad 
\end{align*}
\]

\[
\begin{align*}
\sigma_{A<40} \quad 
\end{align*}
\]

\[
\begin{align*}
\sigma_{A<40} \quad 
\end{align*}
\]
Examples of Partial Plans

R(A,B)  S(B,C)  T(C,D)

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \quad \text{R, S, T} \\
\text{WHERE} & \quad R.B = S.B \quad \text{and} \quad S.C = T.C \quad \text{and} \quad R.A < 40
\end{align*}
\]

Top-down plans

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \quad \text{R, S} \\
\text{WHERE} & \quad R.B = S.B \quad \text{and} \quad R.A < 40
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \quad \text{R, S, T} \\
\text{WHERE} & \quad R.B = S.B \\
\text{AND} & \quad S.C = T.C
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{FROM} \quad \text{R, S, T} \\
\text{WHERE} & \quad R.B = S.B \quad \text{and} \quad S.C = T.C
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{R.A, T.D} \\
\text{FROM} & \quad \text{R, S, T} \\
\text{WHERE} & \quad R.B = S.B \quad \text{and} \quad S.C = T.C
\end{align*}
\]

\[
\begin{align*}
\sigma_{A<40}
\end{align*}
\]
Plan Enumeration Algorithms

• Dynamic programming
  – Classical algorithm [1979]
  – Limited to joins: *join reordering algorithm*
  – Bottom-up

• Rule-based algorithm
  – Database of rules (=algebraic laws)
  – Usually: dynamic programming
  – Usually: top-down
Dynamic Programming

Originally proposed in System R [1979]

• Only handles single block queries:

```sql
SELECT list
FROM   R1, ..., Rn
WHERE cond₁ AND cond₂ AND ... AND condₖ
```

• Heuristics: selections down, projections up
• Dynamic programming: *join reordering*
Join Trees

• $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$

• Join tree:

```
        1
       /\  /
      2// 3
       \  /
         4
```

• A plan = a join tree
• A partial plan = a subtree of a join tree
Types of Join Trees

• Left deep:
Types of Join Trees

• Bushy:

```
   /\    /
  /   \ /   \
R3----R2----R4
     /\    /\  \
R1----R5--R4  \
```

R1, R2, R3, R4, R5
Types of Join Trees

• Right deep:

```
           v
          /\  \
         /   \  /
        /     \ /
       /       \
      R3      R1
           /\  \
          /   \  /
         /     \ /
        /       \
       R5      R2
           /\  \
          /   \  /
         /     \ /
        /       \
     R4
```
Dynamic Programming

Join ordering:

• Given: a query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$

• Find optimal order

• Assume we have a function cost() that gives us the cost of every join tree
Dynamic Programming

• For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ compute the following:
  – $\text{Size}(Q) = \text{the estimated size of } Q$
  – $\text{Plan}(Q) = \text{a best plan for } Q$
  – $\text{Cost}(Q) = \text{the estimated cost of that plan}$
Dynamic Programming

• **Step 1:** For each \( \{R_i\} \) do:
  - \( \text{Size}(\{R_i\}) = B(R_i) \)
  - \( \text{Plan}(\{R_i\}) = R_i \)
  - \( \text{Cost}(\{R_i\}) = (\text{cost of scanning } R_i) \)
Dynamic Programming

• **Step 2:** For each $Q \subseteq \{R_1, \ldots, R_n\}$ of cardinality $i$ do:
  
  – $\text{Size}(Q) = \text{estimate it recursively}$
  – For every pair of subqueries $Q', Q''$ s.t. $Q = Q' \cup Q''$
    
    compute $\text{cost}(\text{Plan}(Q') \bowtie \text{Plan}(Q''))$
    
    • $\text{Cost}(Q) = \text{the smallest such cost}$
    • $\text{Plan}(Q) = \text{the corresponding plan}$
Dynamic Programming

• **Step 3**: Return Plan(\{R_1, \ldots, R_n\})
Example

To illustrate, we will make the following simplifications:

- **Cost**\((P_1 \Join P_2)\) = Cost\((P_1)\) + Cost\((P_2)\) + size(intermediate result(s))
  - Size(intermediate result(s)) =
    - If \(P_1\) = a join, then the size of the intermediate result is size\((P_1)\), otherwise the size is 0
    Similarly for \(P_2\)

- Cost of a scan = 0
Example

- $R \Join S \Join T \Join U$
- Number of tuples: 2000, 5000, 3000, 1000
- Size estimation: $T(A \Join B) = 0.01 \times T(A) \times T(B)$
<table>
<thead>
<tr>
<th>Subquery</th>
<th>Size</th>
<th>Cost</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSTU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subquery</td>
<td>Size</td>
<td>Cost</td>
<td>Plan</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>RS</td>
<td>100k</td>
<td>0</td>
<td>RS</td>
</tr>
<tr>
<td>RT</td>
<td>60k</td>
<td>0</td>
<td>RT</td>
</tr>
<tr>
<td>RU</td>
<td>20k</td>
<td>0</td>
<td>RU</td>
</tr>
<tr>
<td>ST</td>
<td>150k</td>
<td>0</td>
<td>ST</td>
</tr>
<tr>
<td>SU</td>
<td>50k</td>
<td>0</td>
<td>SU</td>
</tr>
<tr>
<td>TU</td>
<td>30k</td>
<td>0</td>
<td>TU</td>
</tr>
<tr>
<td>RST</td>
<td>3M</td>
<td>60k</td>
<td>(RT)S</td>
</tr>
<tr>
<td>RSU</td>
<td>1M</td>
<td>20k</td>
<td>(RU)S</td>
</tr>
<tr>
<td>RTU</td>
<td>0.6M</td>
<td>20k</td>
<td>(RU)T</td>
</tr>
<tr>
<td>STU</td>
<td>1.5M</td>
<td>30k</td>
<td>(TU)S</td>
</tr>
<tr>
<td>RSTU</td>
<td>30M</td>
<td>60k+50k=110k</td>
<td>(RT)(SU)</td>
</tr>
</tbody>
</table>
Reducing the Search Space

- Left-linear trees v.s. Bushy trees
- Trees without cartesian product

Example: \( R(A,B) \Join S(B,C) \Join T(C,D) \)

Plan: \((R(A,B) \Join T(C,D)) \Join S(B,C)\) has a cartesian product
  – most query optimizers will not consider it
Dynamic Programming: Summary

• Handles only join queries:
  – Selections are pushed down (i.e. early)
  – Projections are pulled up (i.e. late)

• Takes exponential time in general, BUT:
  – Left linear joins may reduce time
  – Non-cartesian products may reduce time further
Rule-Based Optimizers

- **Extensible** collection of rules
  - Rule = Algebraic law with a direction
- Algorithm for firing these rules
  - Generate many alternative plans, in some order
  - Prune by cost

- Volcano (later SQL Sever)
- Starburst (later DB2)
Completing the Physical Query Plan

• Choose algorithm for each operator
  – How much memory do we have?
  – Are the input operand(s) sorted?

• Access path selection for base tables

• Decide for each intermediate result:
  – To materialize
  – To pipeline
Access Path Selection

- **Access path**: a way to retrieve tuples from a table
  - A file scan
  - An index *plus* a matching selection condition

- Index matches selection condition if it can be used to retrieve just tuples that satisfy the condition
  - Example: `Supplier(sid,sname,scity,sstate)`
  - B+-tree index on `(scity,sstate)`
    - matches `scity='Seattle'`
    - does not match `sid=3`, does not match `sstate='WA'`
Access Path Selection

- Supplier(sid, sname, scity, sstate)

- Selection condition: $\text{sid} > 300 \land \text{scity} = 'Seattle'$

- Indexes: B+-tree on $\text{sid}$ and B+-tree on $\text{scity}$

- Which access path should we use?

- We should pick the most selective access path
Access Path Selectivity

• Access path selectivity is the number of pages retrieved if we use this access path
  – Most selective retrieves fewest pages

• As we saw earlier, for equality predicates
  – Selection on equality: $\sigma_{a=v}(R)$
  – $V(R, a) = \# \text{ of distinct values of attribute } a$
  – $1/V(R,a)$ is thus the reduction factor
  – Clustered index on a: cost $B(R)/V(R,a)$
  – Unclustered index on a: cost $T(R)/V(R,a)$
  – (we are ignoring I/O cost of index pages for simplicity)
Materialize Intermediate Results Between Operators

HashTable ← S
repeat
  read(R, x)
  y ← join(HashTable, x)
  write(V1, y)

HashTable ← T
repeat
  read(V1, y)
  z ← join(HashTable, y)
  write(V2, z)

HashTable ← U
repeat
  read(V2, z)
  u ← join(HashTable, z)
  write(Answer, u)
Materialize Intermediate Results Between Operators

Question in class

Given B(R), B(S), B(T), B(U)

• What is the total cost of the plan?
  – Cost =

• How much main memory do we need?
  – M =
Pipeline Between Operators

Pipeline

HashTable1 ← S
HashTable2 ← T
HashTable3 ← U
repeat
  read(R, x)
  y ← join(HashTable1, x)
  z ← join(HashTable2, y)
  u ← join(HashTable3, z)
write(Answer, u)
Pipeline Between Operators

Question in class

Given B(R), B(S), B(T), B(U)

• What is the total cost of the plan?
  – Cost =

• How much main memory do we need?
  – M =
Pipeline in Bushy Trees
Example

• Logical plan is:

• Main memory M = 101 buffers
Example

M = 101

Naïve evaluation:
• 2 partitioned hash-joins
• Cost $3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k$
Example

\[ M = 101 \]

\begin{itemize}
\item Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
\item Step 2: hash S on x into 100 buckets; to disk
\item Step 3: read each \( R_i \) in memory (50 buffer) join with \( S_i \) (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we \textit{pipeline}
\item Cost so far: \( 3B(R) + 3B(S) \)
\end{itemize}
Example

M = 101

Continuing:

• How large are the 50 buckets on y? Answer: k/50.
• If k <= 50 then keep all 50 buckets in Step 3 in memory, then:
• Step 4: read U from disk, hash on y and join with memory
• Total cost: 3B(R) + 3B(S) + B(U) = 55,000
Example

M = 101

Continuing:
- If 50 < k <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size k/50 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: $3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k$
Example

M = 101

Continuing: 5,000 blocks 10,000 blocks
• If k > 5000 then materialize instead of pipeline
• 2 partitioned hash-joins
• Cost $3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k$
Outline

• Search space
• Algorithms for enumerating query plans
  • Estimating the cost of a query plan
Computing the Cost of a Plan

• Collect statistical summaries of stored data

• Estimate size in a bottom-up fashion

• Estimate cost by using the estimated size
Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)
- Collection approach: periodic, using sampling
Size Estimation

Estimating the size of a projection

• Easy: $T(\Pi_L(R)) = T(R)$

• This is because a projection doesn’t eliminate duplicates
Size Estimation for Selection

Estimating the size of a selection

- \( S = \sigma_{A=c}(R) \)
  - \( T(S) \) can be anything from 0 to \( T(R) - V(R,A) + 1 \)
  - Estimate: \( T(S) = \frac{T(R)}{V(R,A)} \)
  - When \( V(R,A) \) is not available, estimate \( T(S) = \frac{T(R)}{10} \)

- \( S = \sigma_{A<c}(R) \)
  - \( T(S) \) can be anything from 0 to \( T(R) \)
  - Estimate: \( T(S) = \frac{(c - \text{Low}(R, A))}{(\text{High}(R,A) - \text{Low}(R,A))}T(R) \)
  - When \( \text{Low}, \text{High} \) unavailable, estimate \( T(S) = \frac{T(R)}{3} \)
Size Estimation for Selection

What if we have an index on multiple attributes?
• Example selection $S = \sigma_{a=v_1 \land b=v_2}(R)$

How to compute the selectivity?
• Assume attributes are independent
• $T(S) = T(R) / (V(R,a) \times V(R,b))$
Example

• Selection condition: \( \text{sid} > 300 \land \text{scity} = \text{‘Seattle’} \)
  – Index I1: B+-tree on sid clustered
  – Index I2: B+-tree on scity unclustered

• Let’s assume
  – \( V(\text{Supplier}, \text{scity}) = 20 \)
  – \( \text{Max}(\text{Supplier}, \text{sid}) = 1000, \text{Min}(\text{Supplier}, \text{sid}) = 1 \)
  – \( B(\text{Supplier}) = 100, T(\text{Supplier}) = 1000 \)

• Cost I1: \( B(R) \times (\text{Max-v})/(\text{Max-Min}) = 100 \times 700/999 \approx 70 \)
• Cost I2: \( T(R) \times 1/V(\text{Supplier}, \text{scity}) = 1000/20 = 50 \)
Size Estimation for Join

Estimating the size of a natural join, $R \bowtie_A S$

- When the set of $A$ values are disjoint, then $T(R \bowtie_A S) = 0$
- When $A$ is a key in $S$ and a foreign key in $R$, then $T(R \bowtie_A S) = T(R)$
- When $A$ has a unique value, the same in $R$ and $S$, then $T(R \bowtie_A S) = T(R) \cdot T(S)$

Estimation seems hopelessly hard!
Size Estimation for Join

Assumptions:

• **Containment of values**: if \( V(R,A) \leq V(S,A) \), then the set of A values of R is included in the set of A values of S
  
  – Note: this indeed holds when A is a foreign key in R, and a key in S

• **Preservation of values**: for any other attribute B, \( V(R \bowtie_A S, B) = V(R, B) \) (or \( V(S, B) \))
Size Estimation for Join

Assume $V(R,A) \leq V(S,A)$

- Then each tuple $t$ in $R$ joins some tuple(s) in $S$
  - How many?
  - On average $T(S)/V(S,A)$
  - $t$ will contribute $T(S)/V(S,A)$ tuples in $R \bowtie_A S$

- Hence $T(R \bowtie_A S) = T(R) \cdot T(S) / V(S,A)$

In general: $T(R \bowtie_A S) = T(R) \cdot T(S) / \max(V(R,A),V(S,A))$
Size Estimation for Join

Example:

- \( T(R) = 10000, \quad T(S) = 20000 \)
- \( V(R,A) = 100, \quad V(S,A) = 200 \)
- How large is \( R \bowtie_A S \)?

Answer: \( T(R \bowtie_A S) = \frac{10000 \times 20000}{200} = 1M \)
Size Estimation for Join

Joins on more than one attribute:

• $T(R \bowtie_{A,B} S) =$

$$T(R) \cdot T(S)/(\max(V(R,A),V(S,A)) \cdot \max(V(R,B),V(S,B)))$$
Computing Cost of an Operator

• The cost of executing an operator depends
  – On the operator implementation
  – On the input data

• We learned how to compute this in the previous lecture, so we do not repeat it here
Histograms

• Statistics on data maintained by the RDBMS

• Makes size estimation much more accurate (hence, cost estimations are more accurate)
Histograms

Employee(ssn, name, salary, phone)

- Maintain a histogram on salary:

<table>
<thead>
<tr>
<th>Salary:</th>
<th>0..20k</th>
<th>20k..40k</th>
<th>40k..60k</th>
<th>60k..80k</th>
<th>80k..100k</th>
<th>&gt; 100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>200</td>
<td>800</td>
<td>5000</td>
<td>12000</td>
<td>6500</td>
<td>500</td>
</tr>
</tbody>
</table>

- \( T(\text{Employee}) = 25000 \), but now we know the distribution
Histograms

Employee(ssn, name, salary, phone)

- **Eqwidth**

<table>
<thead>
<tr>
<th>Salary</th>
<th>0..20</th>
<th>20..40</th>
<th>40..60</th>
<th>60..80</th>
<th>80..100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>2</td>
<td>104</td>
<td>9739</td>
<td>152</td>
<td>3</td>
</tr>
</tbody>
</table>

- **Eqdepth**

<table>
<thead>
<tr>
<th>Salary</th>
<th>0..44</th>
<th>44..48</th>
<th>48..50</th>
<th>50..56</th>
<th>55..100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
</tr>
</tbody>
</table>
Example

Employee(ssn, name, salary, phone)

<table>
<thead>
<tr>
<th>Salary</th>
<th>0..44</th>
<th>44..48</th>
<th>48..50</th>
<th>50..56</th>
<th>55..100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
</tr>
</tbody>
</table>

Estimate the size of: \( S = \sigma_{\text{salary} \geq 46 \text{ and } \text{salary} \leq 70}(\text{Employee}) \)
Example

**Employee(ssn, name, salary, phone)**

<table>
<thead>
<tr>
<th>Salary</th>
<th>0..44</th>
<th>44..48</th>
<th>48..50</th>
<th>50..56</th>
<th>55..100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuples</td>
<td>1800</td>
<td>2000</td>
<td>2100</td>
<td>2200</td>
<td>1900</td>
</tr>
</tbody>
</table>

Estimate the size of:  \( S = \sigma_{\text{salary} \geq 46 \text{ and salary} \leq 70}(\text{Employee}) \)

Answer: \( T(S) = 2000 \times \frac{3}{4} + 2100 + 2200 + 1900 \times \frac{16}{46} \)
Summary of Query Optimization

• Three parts:
  – search space, algorithms, size/cost estimation

• This lecture discussed some of the issues
  – Lecture has more material than either textbook, however:
    – You won’t be able to write an optimizer tomorrow!
  – There is no good text on rule-based optimizer