Warm-up Exercise

(See exercise sheet. You can start before class.)

Answer: a reduction factor, which ranges from 0.0 – 1.0, tells us how many tuples remain after a relational operator is applied.

Goals for Today

• Explore the search space explosion for alternate query plans

• Understand the dynamic programming approach to exploring the (large!) space of query plans

• Reason about the heuristics used by the System R query optimizer to prune the space
  – Discuss some of the corners cut by query optimization algorithms like the System R approach

Query Optimizer algorithm

• Goal: given a a query, the optimizer wants to
  – Enumerate query plans to consider
  – Compare plans and choose the “best” one

• Algorithm
  – Step 1: consider a set of possible plans
  – Step 2: estimate cost for each plan
  – Step 3: choose the plan with lowest cost
Logical Transformations:
Equivalent Relational Algebra Expressions

- Can write the same query multiple ways!
  - These alternate versions are akin to different possible logical query plans

- Good rules of thumb:
  - “Push” down selections
  - Avoid cross-products

Good rules of thumb:
- “Push” down selections
- Avoid cross-products

Selections:
\[ \sigma_{c_1 \land \ldots \land c_n} (R) \equiv \sigma_{c_1} (\ldots \sigma_{c_n} (R)) \quad \text{(Cascade)} \]
\[ \sigma_{c_1} (\sigma_{c_2} (R)) \equiv \sigma_{c_2} (\sigma_{c_1} (R)) \quad \text{(Commute)} \]

Projections:
\[ \pi_{a_1} (R) \equiv \pi_{a_1} (\ldots (\pi_{a_n} (R))) \quad \text{(Cascade)} \]
(if \( a_n \) includes \( a_n \) includes \ldots \( a_i \))

A projection could commute with a selection, e.g.,
\[ \pi_a (\sigma_c (R)) \equiv \sigma_c (\pi_a (R)) \quad \text{... if condition } c \text{ acts only on attributes in } a \]

R.A. Equivalences:

- **Joins:**
  \[ (R \bowtie S) \equiv (S \bowtie R) \quad \text{(Commutative)} \]
  \[ R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \quad \text{(Associative)} \]

If theta join, join condition must involve correct relations

These mean we can switch join outer/inner relations and can do joins in any order!

Selection between attributes of the two arguments of a cross-product converts cross-product to a join:
\[ \sigma_{R,a=S,b} (R \times S) \equiv (R \bowtie_{R,a=S,b} S) \]

R.A. Equivalences: Select & Project

- **Selection Push:** selection on attributes of \( R \) commutes with \( R \bowtie S \):
  \[ \sigma_c (R \bowtie S) \equiv \sigma_c (R) \bowtie S \]

- **Projection Push:** A projection applied to join of \( R \) and \( S \) can be pushed before the join by:
  - retaining only attributes of \( R \) and \( S \) needed for the join,
  - or are kept by the projection
\[ \pi_{R,a,S,b} (R \bowtie_{R,a=S,b} S) \equiv (\pi_{R,a} (R)) \bowtie_{R,a=S,b} (\pi_{S,b} (S)) \]
Exercise 2-3

\[ \pi_{R,c} \left( \sigma_{R.a=2 \land R.a=S.c} (R \times S) \right) \]

- Convert cross-product to join with \( R.a = S.c \)
- Commute the select condition \( R.a > 2 \) with join
- Note: cannot push projection \( R.c \) before join
  - But could cascade the projection: project \( R.a, c \) before join, then project \( R.c \) after select

- Joining Boats and Sailors first would yield a lot of tuples, since this would become a cross-product!

Enumeration of Alternative Plans

- There are two main cases:
  - **Single-relation** plans (unary operators only)
  - **Multiple-relation** plans

- For unary operators:
  - For a scan, each available access path (sequential scan / index) is considered; one with the least *estimated* cost is chosen
  - Consecutive Scan, Select, Project and Aggregate operations can be typically pipelined

Exercise 4: Plan Enumeration (Single Relation)

- Take advantage of the index on \(<\text{age, rating}>\) for an index-only query plan!
  - Already sorted on age
  - Can pipeline into Aggregate operator to get average rating per age group

Physical DB Design

- Query optimizer does what it can to use indexes, clustering, and operator implementations

- Database Administrator (DBA) is expected to set up physical design well
  - Consider which indexes to create

**Good DBAs understand query optimizers very well!**
Enumerating Multi-Relation Plans

- Suppose we have \( N \) relations
  - Let’s ignore the space of different join algorithms for a moment
  - Recall: associative and commutative rules mean we can apply joins in any order

- How many join orders? Example: \( N=3 \), \( \{A,B,C\} \)
  - How many tree shapes?
  - Given a tree shape, how many leaf orderings?

Exercise 5: Join Orders

- Leaf orderings given a shape? \( N! \)

- Tree shapes, for a fixed ordering of 4 relations
  - 1 left-deep and linear
  - 1 right-deep and linear
  - 1 bushy
  - 2 linear

Number of Join Orders

- Leaf permutations: \( n! \)
- Tree shapes: Catalan numbers

\[
C(n) = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}
\]

- Join orders(\( n \)) = \( n! \times C(n-1) \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>Join orders(( n ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>1680</td>
</tr>
<tr>
<td>6</td>
<td>30,240</td>
</tr>
<tr>
<td>7</td>
<td>665,280</td>
</tr>
<tr>
<td>8</td>
<td>17,297,280</td>
</tr>
<tr>
<td>9</td>
<td>518,918,400</td>
</tr>
<tr>
<td>10</td>
<td>17,643,225,600</td>
</tr>
</tbody>
</table>

Dynamic Programming Approach

- Brute-force enumeration approach does not scale

- Observation: within the space of all possible plans, many plans share a common subplan

\[
A \bowtie B \quad ((A \bowtie B) \bowtie C) \bowtie D
\]

Best plan to join \( A \) and \( B \) can help us find the best plan to join \( A \), \( B \), \( C \), and \( D \)

- Dynamic programming!
  - Cache best results for plans already considered

System R: Plans to Consider

- Fundamental decision in System R: only left-deep join trees considered (1 tree shape)
- Left-deep trees allow us generate all fully pipelined plans
  - Note: Recall not all left-deep trees are fully pipelined (e.g., Sort-Merge join)
- Selections on a relation processed as part of access path, or on-the-fly with JOINs

Why do we care?

More System R heuristics later...

Enumeration: Dynamic Programming (left-deep)

- Query plans differ by:
  - order of the N relations,
  - access method for each relation,
  - and the join method for each join

- Plans are enumerated in $N$ passes, considering subsets of the $N$ relations
- For each subset of relations, retain:
  - Cheapest plan overall (possibly unordered)

Dynamic Programming Pseudocode

R ← set of relations to join (e.g., ABCD)

for $\partial$ in $\{1 \ldots |R|\}$:
  for $S$ in (all length $\partial$ subsets of $R$):
    $\text{optjoin}(S) = (S - a)$ join $a$

// where $a$ is the single relation that minimizes:
// $\text{cost}($optjoin$(S - a)) +$
min. cost to join $(S - a)$ to $a +$
min. access cost for $a$

$\text{optjoin}(S - a)$ is cached from previous iteration
DP: Example (left-deep)

\[
\text{optjoin}(ABCD)
\]

\(\partial = 1\)

- A = best way to access A
  - (e.g. sequential scan or index)
- B = best way to access B
- C = best way to access C
- D = best way to access D

DP: Example (left-deep)

\[
\text{Plan Cache}
\begin{array}{|c|c|c|c|}
\hline
\text{Subplan} & \text{Best choice} & \text{Cost} & \text{Cardinality} \\
\hline
A & \text{index} & 150 & 1000 \\
B & \text{Seq scan} & 600 & 5000 \\
\{A,B\} & \text{BA} & \ldots & \ldots \\
\{B,C\} & \text{BC} & \ldots & \ldots \\
\{A,B,C\} & \text{ACB} & \ldots & \ldots \\
\{B,C,D\} & \text{CBD} & \ldots & \ldots \\
\hline
\end{array}
\]

\(\partial = 2\)

- \{A,B\} = AB or BA
  - (use pre-computed best way to access A and B)
- \{A,C\} = AC or CA
- \{A,D\} = AD or DA
- \{B,C\} = BC or CB
- \{B,D\} = BD or DB
- \{C,D\} = CD or DC

DP: Example (left-deep)

\[
\text{Plan Cache}
\begin{array}{|c|c|c|c|}
\hline
\text{Subplan} & \text{Best choice} & \text{Cost} & \text{Cardinality} \\
\hline
A & \text{index} & 150 & 1000 \\
B & \text{Seq scan} & 600 & 5000 \\
\{A,B\} & \text{BA} & \ldots & \ldots \\
\{B,C\} & \text{BC} & \ldots & \ldots \\
\{A,B,C\} & \text{ACB} & \ldots & \ldots \\
\{B,C,D\} & \text{CBD} & \ldots & \ldots \\
\hline
\end{array}
\]

\(\partial = 3\)

- \{A,B,C\} = remove A, compare plans for \(\{B,C\}\)
- remove B, compare plans for \(\{A,C\}\)
- remove C, compare plans for \(\{A,B\}\)

\{B,C,D\} = ...

\{A,C,D\} = ...

\{A,B,D\} = ...

DP: Example (left-deep)

\[
\text{Plan Cache}
\begin{array}{|c|c|c|c|}
\hline
\text{Subplan} & \text{Best choice} & \text{Cost} & \text{Cardinality} \\
\hline
A & \text{index} & 150 & 1000 \\
B & \text{Seq scan} & 600 & 5000 \\
\{A,B\} & \text{BA} & \ldots & \ldots \\
\{B,C\} & \text{BC} & \ldots & \ldots \\
\{A,B,C\} & \text{ACB} & \ldots & \ldots \\
\{B,C,D\} & \text{CBD} & \ldots & \ldots \\
\hline
\end{array}
\]

\(\partial = 4\)

- \{A,B,C,D\} = remove A, compare plans for \(\{B,C,D\}\)
- remove B, compare plans for \(\{A,C,D\}\)
- remove C, compare plans for \(\{A,B,D\}\)
- remove D, compare plans for \(\{A,B,C\}\)

\{B,C,D\} = ...

\{A,C,D\} = ...

\{A,B,D\} = ...

\{A,B,C\} = ...
DP Algorithm: Complexity (left-deep)

- Time complexity
  - For each pass $k$, consider all subsets of relations of size $k$
    $\rightarrow \text{N choose k subsets}$
  - All subsets for $N$ relations: $2^N$

- For each subset of size $k$, $k$ ways to remove 1 join ($k \leq N$)

  \[
  \text{Time complexity} = O(N2^N)
  \]

Interesting Orders

- The output relation from a given operator could be ordered
  
  \[
  \text{How?}
  \]

- An intermediate result has an “interesting order” if it is returned in order of any of:

  - ORDER BY attributes
  - GROUP BY attributes
  - Join attributes of other joins

System R: Plans Considered (Contd.)

- Only consider left-deep plans

- In DP algorithm, also keep in plan cache cheapest plan for each interesting order of the tuples

- Avoid Cross-products if possible
  - An $i$-$1$ way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE clause have been used up

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an interestingly ordered plan or an additional sorting operator

Small Example (modified from book Ch. 15)

```
SELECT S.sname
FROM  Sailors S, Reserves R
WHERE  S.sid = R.sid
     AND S.rating > 5
     AND R.bid = 100
```

Pass 1:

- Reserves: Clustered B+ tree on bid matches bid=100, and is cheaper than file scan

  - Sailors: B+ tree matches rating>5, not very selective, and index is unclustered, so file scan w/select is likely cheaper. Also, Sailors.rating is not an interesting order.

Pass 2: We consider each Pass 1 plan as the outer:

  - Reserves as outer (B+Tree selection on bid):
    Find lowest-cost join algorithm with Sailors as Inner
  
  - Sailors as outer (File Scan w/select on rating):
    Find lowest-cost join algorithm with Reserves as Inner
Bigger Example (modified from book Ch.15)

Select S.sid, COUNT(*) AS numRedRes
FROM Sailors S, Reserves R, Boats B
AND B.color = "red"
GROUP BY S.sid

Sailors:
- Unclust. B+ on sid
- Clustered B+ tree on bid

Reserves:
- Unclust. B+ on sid
- Clustered Hash on color

Boats:
- Clustered Hash on color

• **Pass 1: Best plan(s) for accessing each relation (Ex. 6)**
  - **Sailors**: Seq File Scan; B+ on sid
  - **Reserves**: Seq File Scan; B+ on bid, B+ on sid
  - **Boats**: Hash on color

  (note: given selection on color, clustered Hash is likely to be cheaper than file scan, so only it is retained)

As discussed in class:
if optimizer can do index-only plans, would not retain seq. file scan on Sailors and Reserves

Bigger Example: Pass 2

- For each of the plans in Pass 1, generate plans joining another relation as the inner (avoiding cross products)
- Consider all join methods and every access path for the inner:
  - File Scan Reserves (outer) with Boats (inner)
  - File Scan Reserves (outer) with Sailors (inner)
  - B+ on Reserves.bid (outer) with Boats (inner)
  - B+ on Reserves.bid (outer) with Sailors (inner)
  - B+ on Reserves.sid (outer) with Boats (inner)
  - B+ on Reserves.sid (outer) with Sailors (inner)
  - File Scan Sailors (outer) with Reserves (inner)
  - B+Tree Sailors.sid (outer) with Reserves (inner)
  - Hash on Boats.color (outer) with Reserves (inner)

- Retain both:
  - cheapest plan for each pair of relations
  - cheapest plan for each interesting order

Bigger Example: Pass 3

• For each of the plans retained from Pass 2, taken as the outer, generate plans for the remaining join
  - e.g.
    Outer= Hash on Boats.color JOIN Reserves
    Inner = Sailors
    Join Method = Index NL using Sailors.sid B+Tree

• Then, add the cost for doing the GROUP BY and aggregate:
  - This is the cost to sort the result by sid, unless it has already been sorted by a previous operator.

• Then, choose the cheapest plan overall

EXPLAIN

• Many DBMSs support a feature called **EXPLAIN**

• Shows query plan the optimizer would choose
  - Use indexes or sequential scan?
  - Join order? Join algorithms?
SQLite Demo: EXPLAIN QUERY PLAN

```
sqlite> .read sailors.sql

sqlite> EXPLAIN QUERY PLAN SELECT * FROM Sailors WHERE age > 40;
selectid  | order  | from | detail
-----------|--------|------|-------
0          | 0      | 0    | SCAN TABLE Sailors
sqlite> CREATE INDEX ageIndex ON Sailors(age);
sqlite> EXPLAIN QUERY PLAN SELECT * FROM Sailors WHERE age > 40;
selectid  | order  | from | detail
-----------|--------|------|-------
0          | 0      | 0    | SEARCH TABLE Sailors USING INDEX ageIndex (age>?)
sqlite> EXPLAIN QUERY PLAN SELECT age FROM Sailors WHERE age > 40;
selectid  | order  | from | detail
-----------|--------|------|-------
0          | 0      | 0    | SEARCH TABLE Sailors USING COVERING INDEX ageIndex (age>?)
sqlite> SELECT * FROM Sailors,Boats,Reserves
   ...> WHERE Sailors.sid = Reserves.sid
   ...> AND Boats.bid = Reserves.bid
   ...> AND Sailors.rating > 5;
selectid  | order  | from | detail
-----------|--------|------|-------
0          | 1      | 0    | SEARCH TABLE Sailor
0          | 2      | 1    | SEARCH TABLE Boats
sqlite> CREATE INDEX ratingIndex ON Sailors(rating);
sqlite> analyze; /* force sqlite to recompute statistics */
sqlite> EXPLAIN QUERY PLAN SELECT * FROM Sailors,Boats,Reserves
   ...> WHERE Sailors.sid = Reserves.sid
   ...> AND Boats.bid = Reserves.bid
   ...> AND Sailors.rating > 5;
selectid  | order  | from | detail
-----------|--------|------|-------
0          | 0      | 0    | SEARCH TABLE Sailors USING INDEX ratingIndex (rating>?)
0          | 1      | 2    | SEARCH TABLE Reserves USING COVERING INDEX sqlite_autoi
0          | 2      | 1    | SEARCH TABLE Boats USING INTEGER PRIMARY KEY (rowid?)
```