CS 133: Databases

Fall 2019
Lec 15 – 10/31
Prof. Beth Trushkowsky

Administrivia

- New regular office hour!
  - Thursdays 3-4pm, starting today
- In-class worksheets
  - Extras hanging in basket hanging outside my office
  - Answers posted inline with slides on course website
- Problem sets answers
  - I will upload "model answer" on Sakai

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

Relational Algebra Equivalences

Selections:

$$\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots\ \sigma_{cn}(R)) \quad \textit{(Cascade)}$$

$$\sigma_{c1}\left(\sigma_{c2}(R)\right) \equiv \sigma_{c2}\left(\sigma_{c1}(R)\right) \quad \textit{(Commute)}$$

Projections:

$$\pi_{a}(R) \equiv \pi_{a1}\left(\ldots\left(\pi_{an}(R)\right)\right) \quad \textit{(Cascade)}$$

(if a_n includes a_{n-1} includes… a_1)

A projection could commute with a selection, e.g.,

$$\pi_{a}\left(\sigma_{c}(R)\right) \equiv \sigma_{c}\left(\pi_{a}(R)\right)$$

… if condition *c* acts only on attributes in *a*

R.A. Equivalences: Joins

Joins:

$$(R \bowtie S) \equiv (S \bowtie R) \quad \textit{(Commutative)}$$

$$R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \quad \textit{(Associative)}$$

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

$$\pi_{R.a,S.b}\left(R \bowtie_{R.a=S.b} S\right) \equiv \left(\pi_{R.a}(R)\right) \bowtie_{R.a=S.b} \left(\pi_{S.b}(S)\right)$$

Exercise 2-3

2. $$\pi_{R.c}\left(\sigma_{R.a>2 \wedge R.a=S.c}(R \times S)\right)$$

- 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

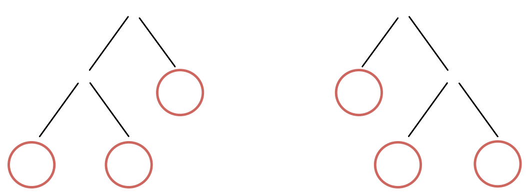
3. Joining Boats and Sailors first would yield a lot of tuples, since this would become a cross-product!

Enumeration of Alternative Plans

- 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*

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?



For both tree shapes, can have 6 orderings of relations in the leaves

Exercise 4: 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 order permutations: n!
- Tree shapes: Catalan numbers

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!}$$

- Join orders(n) = n! * C(n-1)

n	Join orders(n)
1	1
2	2
3	12
4	120
5	1680
6	30,240
7	665,280
8	17,297,280
9	518,918,400
10	17,643,225,600

Source: http://www.necessaryandsufficient.net/2009/06/query-optimisation-plan-space-and-catalan-numbers/

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 ⋈ B **((A ⋈ B) ⋈ C) ⋈ D**

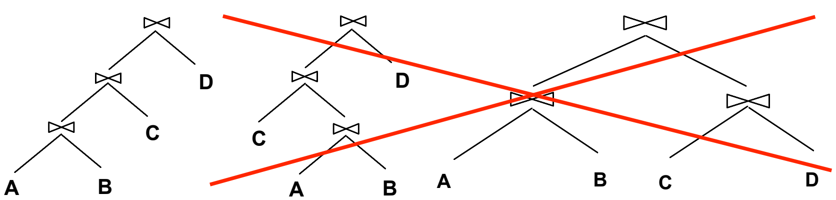
((A ⋈ B) ⋈ D) ⋈ C

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)

Why do we care?

- Selections on a relation processed as part of access path, or on-the-fly with JOINs

More System R heuristics later…

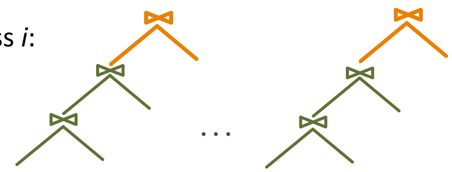
Enumeration: Dynamic Programming (left-deep)

- *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)

We'll also hang onto the cheapest plans for ordered tuples! (Later)

Enumeration: Dynamic Programming (left-deep)

- **Pass 1:** Find best "1-relation" plans for each relation
- **Pass 2:** Find the best ways to join result of each 1-relation plan as outer to another relation.
- …



- **Pass N:** Find best ways to join result of a (N-1)-relation plan as outer to the N'th relation.

Dynamic Programming Pseudocode

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

for ∂ in {1 ...|R|}:
    for S in {all length ∂ subsets of R}:
        optjoin(S) = (S – a) join a

Best way to join all relations in S?

For each *a* in S, try joining it with the best plan for the other S-*a* relations already joined

// where **a** is the single relation that minimizes:
// cost(**optjoin**(S - a)) +
    min. cost to join (S - a) to a +
    min. access cost for a

optjoin(S – a) is cached from previous iteration

DP: Example (left-deep)

Plan Cache

Subplan	Best choice	Cost	Cardinality
A	index	150	1000
B	Seq scan	600	5000
…			

optjoin(ABCD)

∂=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)

Plan Cache

Subplan	Best choice	Cost	Cardinality
A	index	150	1000
B	Seq scan	600	5000

optjoin(ABCD)

∂=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)

Plan Cache

Subplan	Best choice	Cost	Cardinality
A	index	150	1000
B	Seq scan	600	5000
{A,B}	BA	..	..
{B,C}	BC	..	..
…			

optjoin(ABCD)

∂=3

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

{B,C,D} = …

{A,C,D} = …

{A,B,D} = …

DP: Example (left deep)

Plan Cache

Subplan	Best choice	Cost	Cardinality
A	index	150	1000
B	Seq scan	600	5000
{A,B}	BA	..	..
{B,C}	BC	..	..
…			
{A,B,C}	ACB	..	..
{B,C,D}	CBD	..	..

optjoin(ABCD)

∂=4

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

DP Algorithm: Complexity (left-deep)

- Time complexity
  - For each pass *k*, consider all subsets of relations of size *k* → ***N choose k*** subsets
  - All subsets for N relations, less the empty set: **2^N-1**

Power Set: the set of all subsets

  - For each subset of size k, *k* ways to remove 1 join (*k* <= N)

Time complexity = $O(N2^N)$

Interesting Orders

- The output relation from a given operator could be ordered

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

Why would we care?

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

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

Indexes
Reserves:
Clustered B+ tree on *bid*
Sailors:
Unclust B+ tree on *rating*

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 ***sequential 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 (using B+ Tree selection on bid):

Find lowest-cost join algorithm with Sailors as Inner

Sailors as outer (using Seq. File Scan w/selection on rating):

Find lowest-cost join algorithm with Reserves as Inner

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
  - E.g., consider which indexes to create

Good DBAs understand query optimizers very well!

- Many DBMSs support a feature called `EXPLAIN`

Note: Exact syntax varies by DBMS

- Shows query plan the optimizer would choose
  - Use indexes or sequential scan?
  - Join order? Join algorithms?