Overview of Query Optimization

• So far, we studied choices—and their costs—for plans for individual relational operators (selections, projections, joins, …)
• We will now cover more complex queries
Query Optimization: Outline

Given a SQL query, ideally we would:
1. consider all possible execution plans and their estimated cost
2. pick fastest plan and execute it

However:
- Far too many possible execution plans are available
- Should not spend more time finding best plan than executing query with a rough but OK plan

Focus on Two Problems

- Decide which plans we will consider in analysis (not all)
- Design ways of estimating the execution cost of a plan

We will follow the IBM System R approach to query optimization
Highlights of System R Optimizer

• Most widely used; works well for up to ~10 joins
• Cost estimation is an approximate art at best:
  • Based on statistics maintained in database catalogs, to estimate cost of operations and result sizes
  • Based on combination of CPU and I/O costs; we will focus just on I/O costs in our class
• Space of possible execution plans is far too large, so it must be pruned, as we will see
• Each execution plan is represented as a relational algebra tree, with each relational operator in the tree annotated with a choice of implementation algorithm

Cost Estimation for Execution Plan

• Represent execution plan as annotated relational algebra tree
• Estimate cost of each operation in plan tree
• For this, need to estimate size of each result for each operator in tree
Motivating Example

**Sailors** *(sid, sname, rating, age)*

**Boats** *(bid, bname, color)*

**Reserves** *(sid, bid, day)*

One possible relational algebra tree, without annotations (so not a complete plan)

Tuples “flow” from the leaves of the tree up to the root, where the output of the query is produced

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

Motivating Example

**Sailors** *(sid, sname, rating, age)*

**Boats** *(bid, bname, color)*

**Reserves** *(sid, bid, day)*

We can annotate the relational algebra tree to specify an execution plan fully

The left child of the join operator is the outer relation in a nested loops execution, by convention

Cost? Refer to analysis of block nested loops join in previous lecture; \( \sigma \) and \( \pi \) don’t incur further I/Os in this plan: they are evaluated “on the fly” as join tuples flow up the tree

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid = S.sid AND R.bid = 100 AND S.rating > 5
```
Other Plans for Query?

Two main sources of alternative plans:
- Different annotations for same relational algebra tree, with different algorithms for same operators (e.g., “SORT MERGE” instead of “BLOCK NESTED LOOPS” join)
- Different trees (and annotations), with different ordering of operations, or swapping inner and outer relations, but of course still producing the same query results

*Useful principle:* push selections and projections “down” the relational algebra tree (examples on board), to reduce size of intermediate results as early as possible

Cost Analysis of Plans?

- Need to **estimate size of “intermediate” relations**: they are the input of operators higher in relational algebra tree
Joins of Three or More Relations?

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

System R restricts the family of plans it considers to plans with relational algebra trees that:

- don't involve any cross products of relations if possible, to avoid generating very large intermediate relations
- are “left-deep trees,” where the right child of a join operator is never another join operator (i.e., join operators always occur on the left subtree of a node); this restriction reduces the number of trees to consider

Left-Deep Trees (or Plans)

- Can be “pipelined,” without having to materialize on disk the left-side relations (tuples “flow” from the outer relation—on the left—up the tree)
- Still up to \( N! \) (N factorial) relational algebra tree options to consider for an N-way join (N! ways to assign N relations to the N leaves of the only left-deep tree “shape” with N leaves)
Summary

• Several alternative evaluation algorithms for each relational operator
• A query is evaluated by converting it to a relational algebra tree of operators and evaluating the operators in the tree
• Must understand query optimization in order to fully understand the performance impact of a given database design (relations, indexes) on a workload (set of queries)

Two parts to optimizing a query:
• Consider a set of alternative plans
  • Must prune search space; typically, left-deep plans/trees only
  • Must estimate cost of each plan that is considered
    • Size of result and cost for each plan node
    • Key issues: Statistics, indexes, operator implementations

Query optimization studied in detail in CS W4112-Database System Implementation