Introduction to Database Systems

2023-Fall

What is needed for query optimization?

- **Given: A closed set of operators**
  - Relational ops (table in, table out)
  - Physical implementations (of those ops and a few more)

1.Plan space

  - Based on relational equivalences, different implementations

2.Cost Estimation based on

  - Cost formulas
  - Size estimation, in turn based on
    - Catalog information on base tables
    - Selectivity (Reduction Factor) estimation
- Target: To sift through the plan space and find lowest cost option!

Plan Space Review

- For a SQL query, full plan space:
  - All equivalent relational algebra expressions
    - Based on the equivalence rules we learned
  - All mixes of physical implementations of those algebra expressions
- We might prune this space:
  - Selection/Projection pushdown
  - Avoid cartesian products

Plan Space: Too Large, must be pruned.

- Only the space of ***left-deep plans*** is considered.
- Left-deep plans allow output of each operator to be ***pipelined*** into the next operator without storing it in a temporary relation.
- Cartesian products avoided.

Schema for Examples

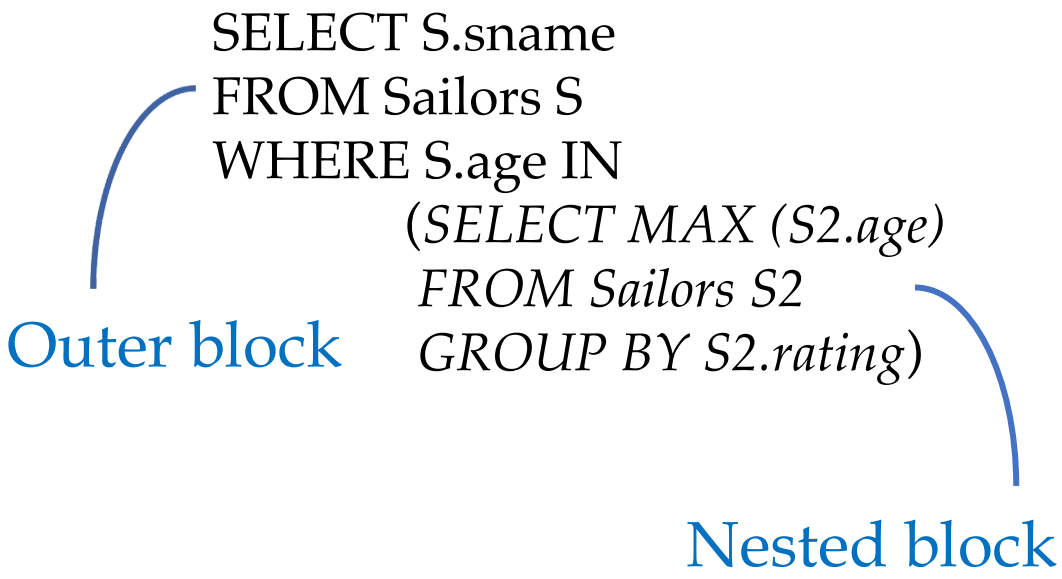
Sailors (*sid*: integer, *sname*: text, *rating*: integer, *age*: float)

Reserves (*sid*: integer, *bid*: integer, *day*: date, *rname*: text)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - 100 distinct bids.
- Sailors:
  - Each tuple is 50 bytes long,
  - 80 tuples per page, 500 pages.
  - 10 ratings, 40,000 sids.

Query Blocks: Units of Optimization

- An SQL query is parsed into a collection of ***query blocks***, and these are optimized one block at a time.
- Nested blocks are usually treated as calls to a subroutine, made once per outer tuple. (This is an over-simplification, but serves for now.)
- For each block, the plans considered are:
  - All *left-deep join trees (*all ways to join the relations one-at-a-time)



Cost Estimation

- For each plan considered, must estimate total cost:
  - Must estimate ***cost*** of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed this for various operators
      - sequential scan, index scan, joins, etc.
  - Must estimate ***size of result*** for each operation in tree!
    - Because it determines downstream input cardinalities!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Statistics and Catalogs

- Need info on relations and indexes involved.
- **Catalogs** typically contain at least:

Statistic	Meaning
NTuples	# of tuples in a table (cardinality)
NPages	# of disk pages in a table
Low/High	min/max value in a column
Nkeys	# of distinct values in a column
IHeight	the height of an index
INPages	# of disk pages in an index

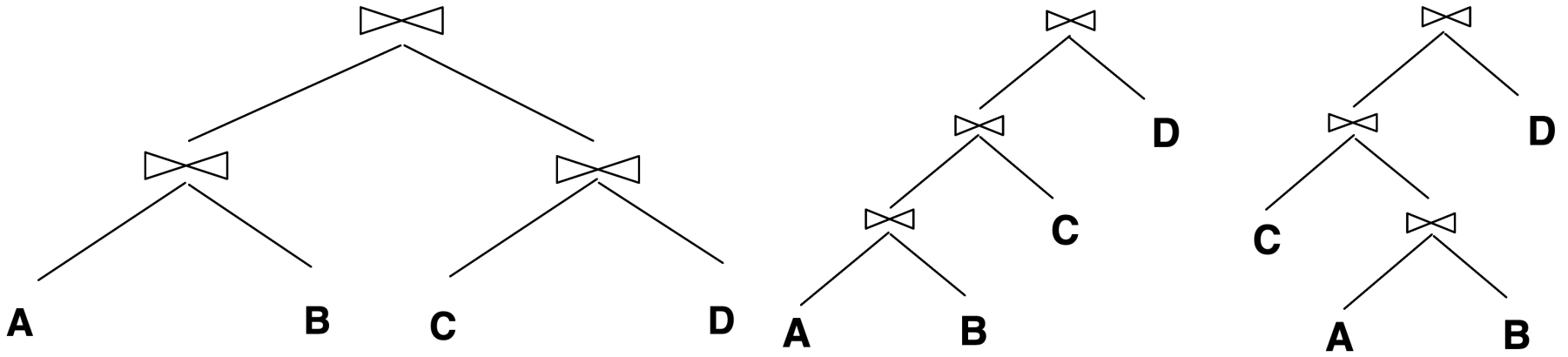
- Catalogs updated periodically.
  - Too expensive to do continuously
  - Lots of approximation anyway, so a little slop here is ok.
- Modern systems do more
  - Esp. keep more detailed statistical information on data values
    - e.g., ***histograms***

Cost Estimates for Single-Relation Plans

- Index I on primary key matches selection:
  - *Cost is Height(I)+1 for a B+ tree, about 1.2 for hash index.*
- Clustered index I matching one or more selects:
  - *(NPages(I)+NPages(R)) * product of RF's of matching selects.*
- Non-clustered index I matching one or more selects:
  - *(NPages(I)+NTuples(R)) * product of RF's of matching selects.*
- Sequential scan of file:
  - *NPages(R).*
- + ***Note:*** *Typically, no duplicate elimination on projections! (Exception: Done on answers if user says DISTINCT.)*

Queries Over Multiple Relations

- Fundamental decision in System R: *only* ***left-deep join trees*** are considered.
- As the number of joins increases, the number of alternative plans grows rapidly; ***we need to restrict the search space.*** Left-deep trees allow us to generate all *fully pipelined* plans.
- Intermediate results not written to temporary files.
  Not all left-deep trees are fully pipelined (e.g., SortMerge join).



Enumeration of Left-Deep Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

Enumeration of Plans (Contd.)

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

- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.

- In spite of pruning plan space, this approach is still exponential in the # of tables.

Cost Estimation for Multi-relation Plans

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- ***Reduction factor (RF)*** associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples * product of all RF's.
- Multi-relation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate