Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
  - Must estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.
- We'll discuss the System R cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

Statistics and Catalogs

- Need information about the relations and indexes involved. ***Catalogs*** typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

- Consider a query block:

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- 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.
  - Implicit assumption that *terms* are independent!
  - Term *col=value* has RF *1/NKeys(I)*, given index I on *col*
  - Term *col1=col2* has RF *1/MAX(NKeys(I1), NKeys(I2))*
  - Term *col>value* has RF *(High(I)-value)/(High(I)-Low(I))*

Histograms

- Key to obtaining good cost and size estimates.
- Come in several flavors:
  - Equi-depth
  - Equi-width
- Which is better?
- Compressed histograms: special treatment of frequent values.

Plans for Single-Relation Queries (Prep for Join ordering)

- **Task:** create a query execution plan for a single Select-project-group-by block.
- **Key idea**: consider each possible ***access path*** to the relevant tuples of the relation. Choose the cheapest one.
- The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are *pipelined* into the aggregate computation).

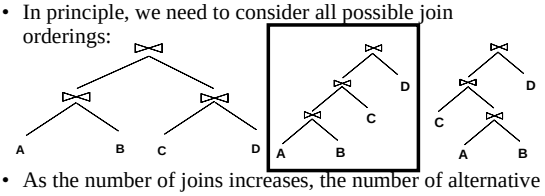
Example

```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```

- If we have an Index on *rating*:
  - (1/NKeys(I)) * NTuples(R) = (1/10) * 40000 tuples retrieved.
  - Clustered index: (1/NKeys(I)) * (NPages(I)+NPages(R)) = (1/10) * (50+500) pages are retrieved (= 55).
  - Unclustered index: (1/NKeys(I)) * (NPages(I)+NTuples(R)) = (1/10) * (50+40000) pages are retrieved.
- If we have an index on *sid*:
  - Would have to retrieve all tuples/pages. With a clustered index, the cost is 50+500.
- Doing a file scan: we retrieve all file pages (500).

Determining Join Order

- In principle, we need to consider all possible join orderings:



- As the number of joins increases, the number of alternative plans grows rapidly; *we need to restrict the search space.*
- System-R: consider *only left-deep join trees*.
  - 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., SM join).

Enumeration of Left-Deep Plans

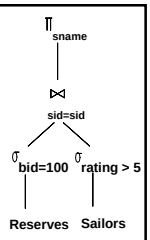
- Naïve approach: n! combinations.
- Principle of optimality: the best plan for the join of $R_1,\ldots R_{n-1}$ will be part of the best plan for the join of $R_1,\ldots,R_n$
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find 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.
- If we want to consider all (bushy) trees, we need only a slight modification to the algorithm.

Example

Sailors:
B+ tree on *rating*
Hash on *sid*
Reserves:
B+ tree on *bid*



- Pass 1: (essentially, access-path selection)
  - *Sailors*: B+ tree matches *rating>5*, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (tuples are in *rating* order).
  - *Reserves*: B+ tree on *bid* matches *bid=100*; cheapest.
- Pass 2: We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
  - e.g., *Reserves as outer*: Hash index can be used to get Sailors tuples that satisfy *sid* = outer tuple's *sid* value.