Size Estimation & Reduction Factors

- Consider a query block:

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

- *Reduction factor (RF) or Selectivity* of each *term*:
  - *Assumption 1*: *uniform distribution of the values*!
  - Term *col=value*: RF = *1/NKeys(I),* given index I on *col*
  - Term *col>value*: RF = *(High(I)-value)/(High(I)-Low(I))*
  - Term *col1=col2*: RF = *1/MAX(NKeys(I1), NKeys(I2))*
- *Max. number of tuples in result* = the product of the cardinalities of relations in the FROM clause.
- *Result cardinality* = Max # tuples * product of all RF's.
  - *Assumption 2*: *terms are independent*!

Size Estimation & Reduction Factors

❖ Consider a query block:

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

❖ *Reduction factor (RF) or Selectivity* of each *term* reflects the impact of the *term* in **reducing** result size.

- *Assumption 1*: *uniform distribution of the values*!
- Term *col=value*: RF = *1/NKeys(I),* given index I on *col*
- Term *col>value*: RF = *(High(I)-value)/(High(I)-Low(I))*
- Term *col1=col2*: RF = *1/MAX(NKeys(I1), NKeys(I2))*
  - *Each value from R with the smaller index I1 has a matching value in S with the larger index I2.*
  - *Values in S are evenly distributed.*
  - *So each R tuple has NTuples(S)/NKeys(I2) matches, a RF of 1/NKeys(I2).*

Cost Estimation for Multi-relation Plans

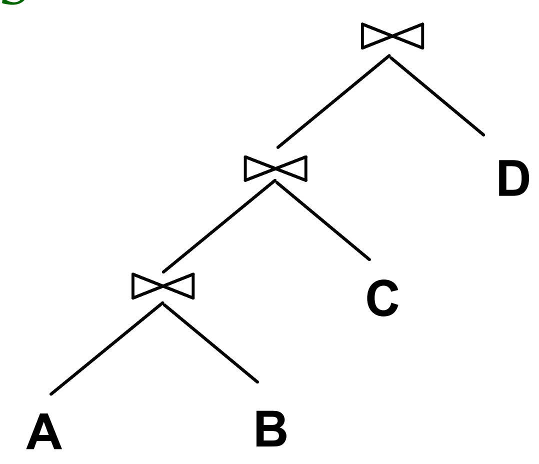
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk

- Consider a query block:
- *Reduction factor (RF)* is associated with each *term*.
- *Max number tuples in result* = the product of the cardinalities of relations in the FROM clause.
- *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 estimate of join cardinality gives us both cost estimate and result size estimate.

Queries Over Multiple Relations

- As the number of joins increases, the number of alternative plans grows rapidly.
- System R: (1) use *only left-deep join trees, where the* inner is a base relation, (2) avoid cartesian products.
  - Allow *pipelined* plans; intermediate results not written to temporary files.
  - Not all left-deep trees are fully pipelined!
    - Sort-Merge join (the sorting phase)
    - Two-phase hash join (the partitioning phase)

Left-deep



Plan space search

- Left-deep join plans differ in:
  - the order of relations,
  - the access path for each relation, and
  - the join method for each join.
- Many of these plans share common prefixes, so don’t enumerate all of them. This is a job for…
- Dynamic Programming

  “a method of solving problems exhibiting the properties of overlapping subproblems and optimal substructure that takes much less time than naive methods.”

Nested Queries With No Correlation

- *Nested query (block)*: a query that appears as an operand of a predicate of the form "expression operator query".
- *Nested query with no correlation*: the nested block does not contain a reference to tuple from the outer.
  - A nested query needs to be evaluated *only once*.
  - The optimizer arranges it to be evaluated before the top level query.

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.rating >
   (SELECT  Avg(rating)
    FROM  Sailors)
```

```
   (SELECT  Avg(rating)
    FROM  Sailors)
```

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.rating > value
```

Nested Queries With Correlation

- Nested query with correlation: the nested block contains a reference to a tuple from the outer.
  - Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
  - The nested block is executed using ***nested iteration***, a tuple-at-a-time approach.

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS
   (SELECT  *
    FROM    Reserves R
    WHERE   R.bid=103
     AND    R.sid=S.sid)
```

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS
   ( …)
```

<u>Nested block to optimize:</u>

```
(SELECT  *
 FROM   Reserves R
 WHERE R.bid =103
   AND   S.sid = outer value)
```

Query Decorrelation

- Implicit ordering of nested blocks means *nested iteration* only.
- The equivalent, non-nested version of the query is typically optimized better, e.g. *hash join* or *sort-merge*.
- *Query decorrelation* is an important task of optimizer.

```
SELECT  S.sname
FROM    Sailors S
WHERE  EXISTS
   (SELECT  *
    FROM     Reserves R
    WHERE  R.bid=103
     AND      R.sid=S.sid)
```

Equivalent non-nested query:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE  S.sid=R.sid
   AND  R.bid=103
```

Summary

- Query optimization is an important task in relational DBMS.
- Must understand optimization in order to 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 only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues*: Statistics, indexes, operator implementations.

Many other research directions

- Extensible query optimizers
- Optimization of expensive predicates
- Multiple-query optimization
- Adaptive query processing