Relational Query Optimization

1

Overview of Query Optimization

- <u>Plan</u>: *Tree of RA ops, with choice of alg for each op.*
  - Each operator typically implemented using a `pull' interface: when an operator is `pulled' for its next output tuple(s), it `pulls' on its input(s) and computes them.
- Two main issues:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
- Ideally: Want to find best plan. Practically: Avoid worst plans!
- We will study the System R approach.

2

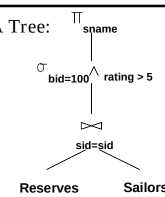
Highlights of System R Optimizer

- Impact:
  - Most widely used currently; works well for < 10 joins.
- Cost estimation: Approximate art at best.
  - Statistics, maintained in system catalogs, used to estimate cost of operations and result sizes.
  - Considers combination of CPU and I/O costs.
- 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 *<u>pipelined</u>* into the next operator without storing it in a temporary relation.
  - Cartesian products avoided when possible.

3

Motivating Example

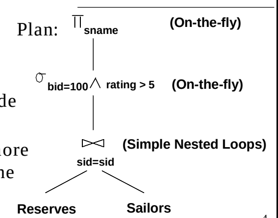
RA Tree:




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

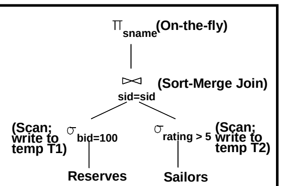
- Cost: 1000 + 1000 * 500 I/Os
- By no means the worst plan!
- Misses several opportunities: selections could have been `pushed' earlier, no use is made of any available indexes, etc.
- *Goal of optimization:* To find more efficient plans that compute the same answer.

Plan:



4

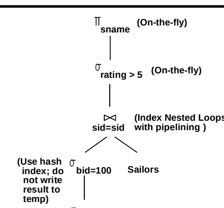
Alternative Plan 1 (No Indexes)



- ***Main difference:*** <u>*push selects.*</u>
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*3*250), merge (10+250)
  - Total: 3560 page I/Os.
- If we used BNL join, join cost = 10+4*250, total cost = 2770.
- If we `push' projections, T1 has only *sid*, T2 only *sid* and *sname*:
  - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.

5

Alternative Plan 2 (With Indexes)



- With clustered index on *bid* of Reserves, we get 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- INL with ***<u>pipelining</u>*** (outer is not materialized).
  - –Projecting out unnecessary fields from outer doesn't help.
- Join column *sid* is a key for Sailors.
  - –At most one matching tuple, unclustered index on *sid* OK.
- Decision not to push *rating>5* before the join is based on availability of *sid* index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple (1000*1.2); total 1210 I/Os.

6

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - 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.

7

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 usually stored.

8

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))*

9

Result Size Estimation (cont'd)

- Selections:
  - Use reduction factors, logic, probability
  - Often use histograms, other statistics
- Projections:
  - Ignore duplicate elimination (done last, if needed)
  - Tuple size is reduced $\Rightarrow$ fewer pages for result
- Joins: like selections, plus:
  - Use candidate key information
  - Should consider "dangling" tuples

10

Relational Algebra Equivalences

- Allow us to choose different join orders and to `push' selections and projections ahead of joins.
- <u>Selections</u>: $\sigma_{c1\wedge\ldots\wedge cn}(R) \equiv \sigma_{c1}(\ldots\sigma_{cn}(R))$ *(Cascade)*

  $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ *(Commute)*
- <u>Projections</u>: $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{an}(R)))$ *(Cascade)*
- <u>Joins</u>: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ *(Associative)*

  $(R \bowtie S) \equiv (S \bowtie R)$ *(Commute)*

11

More Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with $R \bowtie S$. (i.e., $\sigma(R \bowtie S) \equiv \sigma(R) \bowtie S$)
- Similarly, if a projection follows a join R |X| S, we can `push' it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

12

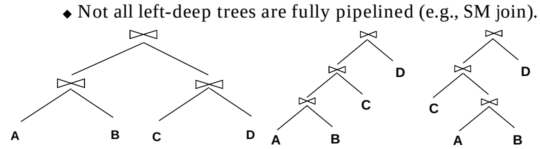
Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans
- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - 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).

13

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., SM join).



14

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 Nth 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.

15

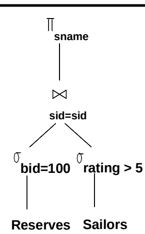
Enumeration of Plans (Contd.)

- 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.

16

Example

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



- Pass1:
  - *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 (because tuples are in *rating* order).
  - *Reserves*: B+ tree on *bid* matches *bid=500*; 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.

17

Summary

- Query optimization is an important task in a relational DBMS.
- Typically optimize 1 "select…" (query block) at a time
- 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.

18

State of the Art (partial snapshot)

- Better histograms (compressed, n-dimensional)
- Run-time (adaptive) query optimization
- Improved buffering / caching techniques
- Performance evaluation and benchmarks
- Cost models for "federated" systems
- More inclusive algebras
  - SQL-92: grouping, aggregates, ordering, etc.
  - SQL-99: object-relational features

19