Overview of Query Evaluation

Chapter 12

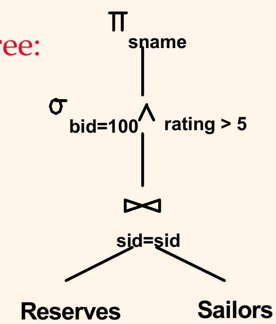
Overview of Query Evaluation

- *Plan: Tree of R.A. ops, with choice of alg for each op.*
  - Each operator is typically implemented using a `pull' interface: when an operator is `pulled' for the next output tuples, it `pulls' on its inputs and computes them.
- Two main issues in query optimization:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for the cheapest (estimated) plan.
  - How is the cost of a plan estimation?
- Ideally: Want to find the best plan. Practically: Avoid worst plans!

Overview of Query Evaluation

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

RA Tree:



Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
  - Must also estimate *size of result* for each operation in tree!
    - Use information about the input relations.

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from a relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 or in reln. 2 (or both).
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*!

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.

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))*

Some Common Techniques

- Algorithms for evaluating relational operators use some simple ideas extensively:
  - Indexing: Can use WHERE conditions to retrieve small set of tuples (selections, joins)
  - Iteration: Sometimes, faster to scan all tuples even if there is an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
  - Partitioning: By using sorting or hashing, we can partition the input tuples.

* *Watch for these techniques as we discuss query evaluation!*

Access Path

- An access path is a method of retrieving tuples:
  - File scan, or index that matches a selection (in the query)
- A *tree index* *matches* (a conjunction of) terms that involve only attributes in a *prefix* of the search key.
  - E.g., Tree index on <*a, b, c*> matches the selection *a*=5 AND *b*=3, and *a*=5 AND *b*>6, but not *b*=3.
- A *hash index* *matches* (a conjunction of) terms that has a term *attribute = value* for every attribute in the search key of the index.
  - E.g., Hash index on <*a, b, c*> matches *a*=5 AND *b*=3 AND *c*=5; but it does not match *b*=3, or *a*=5 AND *b*=3, or *a*>5 AND *b*=3 AND *c*=5.

One Approach to Selections

- Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that do not match the index:
  - *Most selective access path:* An index or file scan that we estimate will require the fewest page I/Os.
  - Terms that match this index reduce the number of tuples *retrieved*; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  - Consider *day*<8/1/2016 AND *bid*=5 AND *sid*=3. A B+ tree index on *day* can be used; then, *bid*=5 and *sid*=3 must be checked for each retrieved tuple. Similarly, a hash index on <*bid, sid*> could be used; *day*<8/1/2016> must then be checked.

Relational Operation - Selection

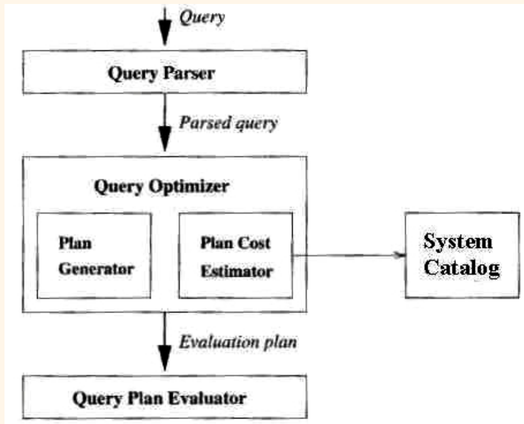
- No indexes => scan the file
- With an index, cost depends on #qualifying tuples, and clustering.
  - *Cost of finding qualifying data entries* (typically small) plus *cost of retrieving records* (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered B+ tree index, cost is little more than 100 I/Os; if unclustered, up to 10000 I/Os!

Projection

```
SELECT  DISTINCT
        R.sid, R.bid
FROM    Reserves R
```

- The expensive part is removing duplicates.
  - SQL systems don't remove duplicates unless the keyword DISTINCT is specified in a query.
- *Sorting Approach*: Sort on <sid, bid> and remove duplicates. (Can optimize this by dropping unwanted information while sorting.)
- *Hashing Approach*: Hash on <sid, bid> to create partitions. Load partitions into memory one at a time and eliminate duplicates.
- If there is an index with both R.sid and R.bid in the search key, may be cheaper to *sort data entries in the index*.

Query Optimizer



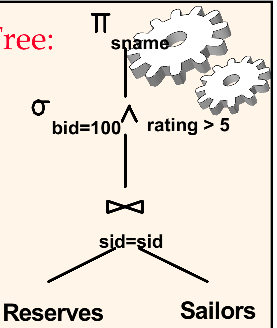
Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- Similar to old schema; *rname* added for variations.
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

Motivating Example

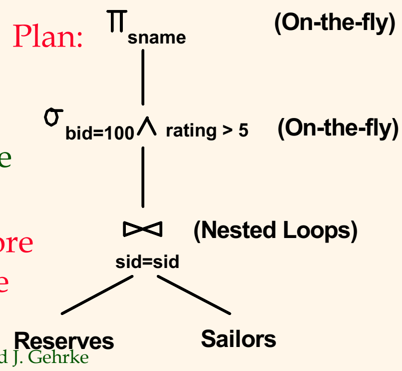
RA Tree:



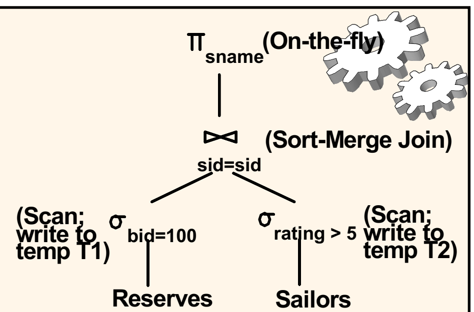

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

- 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:



Alternative Plan



- ***Main difference:*** *push selects.*
- With 5 buffers, cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have one hundred boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250)
  - Total: 4060 page I/Os.
- If we `push' projections, T1 has only *sid*, T2 only *sid* and *sname*:
  - The cost of the join step drops significantly, total < 2000.

Summary

- There are several alternative evaluation algorithms for each relational operator.
- A query is evaluated by converting it to a 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 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.