Review

- Implementation of single Relational Operations
- Choices depend on indexes, memory, stats,...
- Joins
  - Blocked nested loops:
    • simple, exploits extra memory
  - Indexed nested loops:
    • best if 1 rel small and one indexed
  - Sort/Merge Join
    • good with small amount of memory, bad with duplicates
  - Hash Join
    • fast (enough memory), bad with skewed data
Query Optimization Overview

- Query can be converted to relational algebra
- Rel. Algebra converted to tree, joins as branches
- Each operator has implementation choices
- Operators can also be applied in different order!

\[
\text{SELECT S.sname} \\
\text{FROM Reserves R, Sailors S} \\
\text{WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5}
\]

\[
\pi_{(\text{sname})} \sigma_{(\text{bid}=100 \land \text{rating} > 5)} \text{(Reserves } \llp \text{Sailors)}
\]

Query Optimization Overview (cont)

- **Plan:** Tree of R.A. ops, with choice of algorithm for each op.
  - Each operator 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:**
    - 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.
  - **Reality:** Avoid worst plans!
Query Sub-System Architecture

- Query Parser
- Query Optimizer
  - Plan Generator
  - Plan Cost Estimator
- Catalog Manager
- Query Plan Evaluator

Schema for Examples

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

- As seen in previous two lectures...
- 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

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

RA Tree:

```
      T
  σ
  bid=100  rating > 5
      σ
  sid=sid
```

Reserves  Sailors

Simple Nested loops

Plan:

```
      T
  σ
  bid=100  rating > 5
      σ
  sid=sid
```

Sailors  Reserves

(On-the-fly)

(On-the-fly)

(Simple Nested loops)

- Cost: $500 + 500 \times 1000$ 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.
Alternative Plans 1
(No Indexes)

- Main difference: push selects.
- 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 have 10 ratings).
  - Sort T1 ($2^2*10$), sort T2 ($2^2*250$), merge ($10+250$)
  - Total: 3560 page I/Os.
- If use BNL join, join = $10+4*250$, total cost = 2770.
- Can also `push' projections, but must be careful!
  - T1 has only sid, T2 only sid, sname:
  - T1 fits in 3 pgs, cost of BNL under 250 pgs, total < 2000.

Alt Plan 2: Indexes, Pipelining

- With clustered index on bid of Reserves, we get $100,000/100 = 1000$ tuples on $1000/100 = 10$ pages.
- INL with pipelining (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); then, for each, must get matching Sailors tuple (1000*1.2); total 1210 I/Os.
What is needed for optimization?

- Iterator Interface
- Cost Estimation
- Statistics and Catalogs
- Size Estimation and Reduction Factors

**Iterator Interface**

- **A note on implementation:**

  Relational operators at nodes support uniform *iterator* interface:

  ```plaintext
  Open(), get_next(), close()
  ```

  Alternative if pipelining (i.e. a "push"-based approach.

  Can combine using special operators.
Summary

- **Query optimization is an important task in a 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.