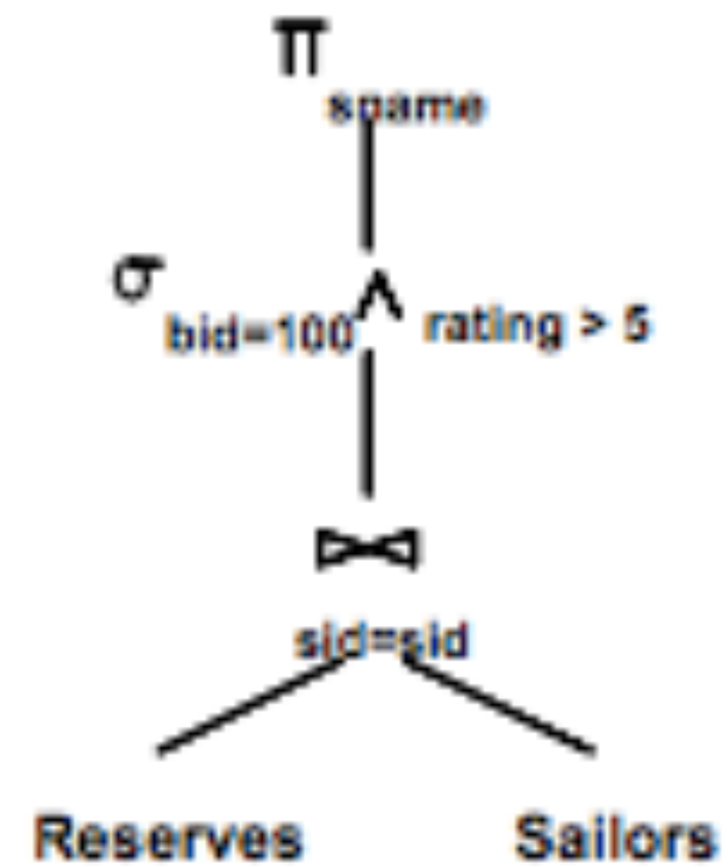
CS 186 Section 9: Query Optimization

Vikram Sreekanti

What is Query Optimization?

- Convert a query to a tree of relational algebra operators.
- Figure out the optimal order — certain operators can be reordered.

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



Query Optimization Overview

There are three main questions for query optimization:

1. What plans are considered?
2. How do we estimate the cost of a plan?
3. How do we search the plan space?

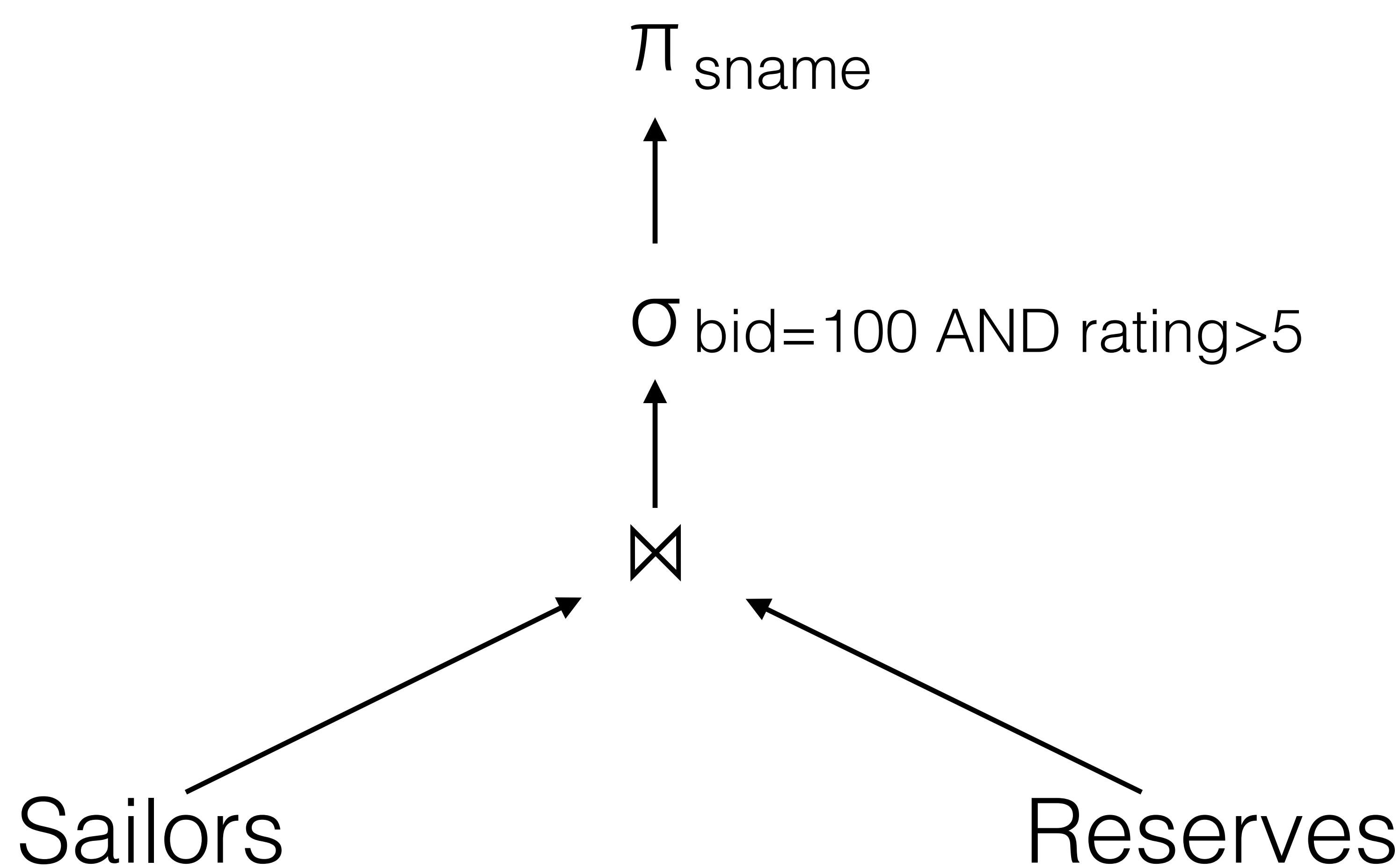
Query Optimization Overview

A great idea for optimization: Push your selections down!
Reserves:
  Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  Assume there are 100 boats
Sailors:
  Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  Assume there are 10 different ratings
Assume we have 5 pages in our buffer pool!

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

Query Optimization Overview

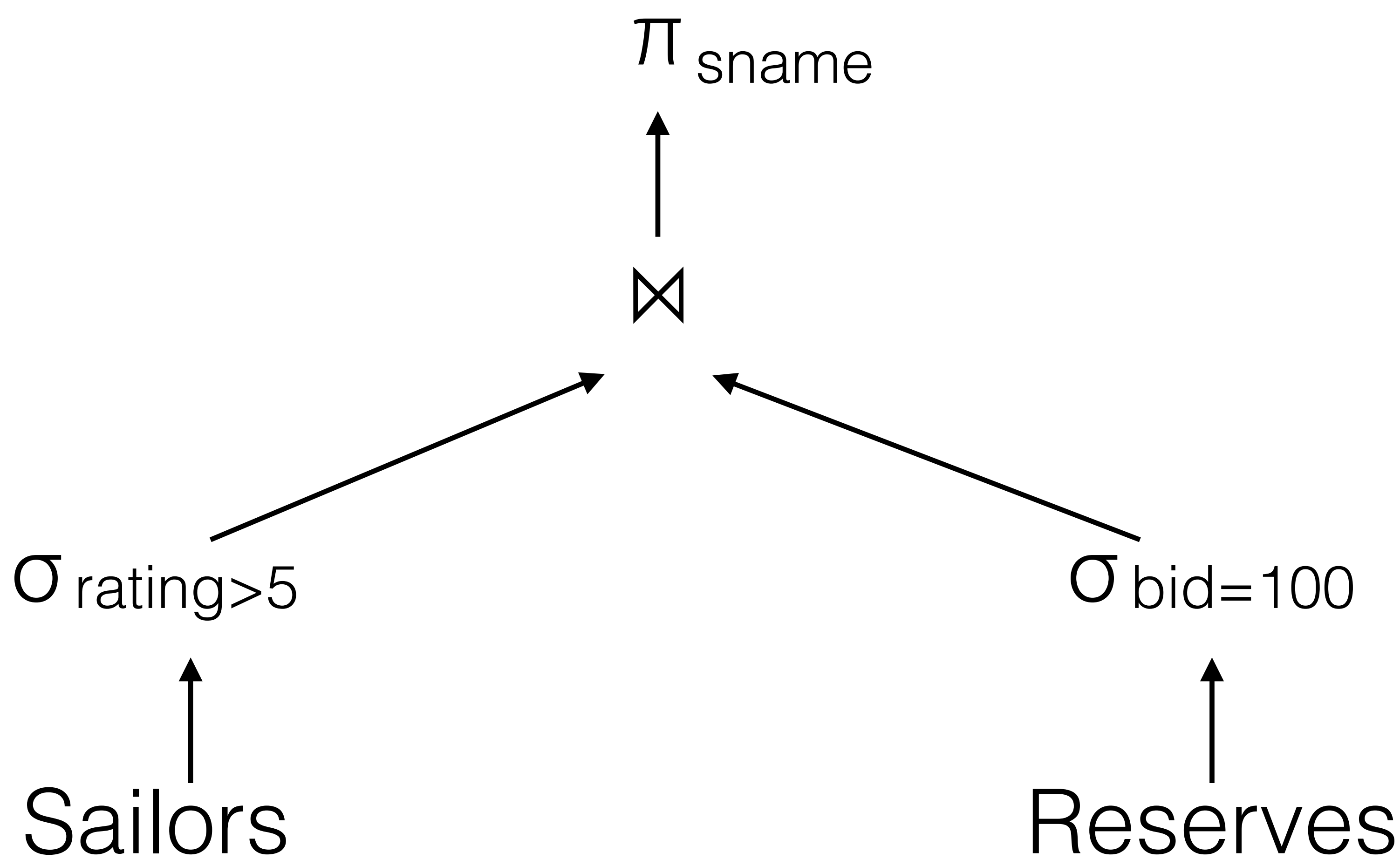
A great idea for optimization: Push your selections down!



COST: 500,500 IOs

Query Optimization Overview

A great idea for optimization: Push your selections down!



COST: 4,250 IOs

Query Optimization Overview

As you can imagine, there are a very large number of plans… so how do we possibly approach query optimization?

There are many, many solutions.

We are going to focus on System-R (aka Selinger-style optimization).

Selinger-Style Optimization

- Works well for up to 10-15 joins — beyond that, the dynamic programming space becomes too large.
  - However, most queries won't even be close to 10-15 joins, so this isn't that big of a constraint.

Selinger-Style Optimization

1. Plan Space is too large — you must prune it!
   a. Ignore plans with overpriced subtrees.
   b. Only consider left-deep plans.
      i. What does this mean?
   c. Put off Cartesian products.
2. Cost Estimation.
   a. Emphasis on estimate — very inaccurate.
   b. Use statistics (histograms); considers IOs and CPU cost. We will ignore CPU cost; only consider IOs.
3. Search Algorithm — use a Dynamic Programming algorithm.

Plan Space

- Convert SQL to relational algebra.
- Use relational algebra equivalences to find possible reordering of plans — helps find the cheapest plan.

Cost Estimation

- Estimate the cost of each operation in the tree.

- Estimate the cost of the output of each operation in the tree because it affects the size of the input to the next operator up the tree.

- Assuming uniform distributions and independent predicates, the selectivities of predicates can be applied in any order to determine the reduction factor .

Histograms

- We use histograms — maintained as statistics in database catalog — to estimate selectivity.
- Equiwidth: range of each bucket is the same size, variable number of values in each bucket.
- Equidepth: values in each bucket is (roughly) the same, variable range to accommodate equal-sized buckets.

Searching the Plan Space

- **`Pass 1:`**
  - Start by enumerating all possible single-table access methods and choose the best access method for each table (as well as ones with interesting orders).
- **`Pass 2 through n:`**
  - Find the best way to join the result of a plan of `size (i - 1)` with another relation.
  - For each subset of relations, retain the cheapest plan *and* interesting orders.

Query Optimization Exercises

Tables:

Kitties: kid[int], cuteness [1-10], owner [10 distinct]):
100 pages, 400 tuples

Puppies: (pid [int], yappiness [1-10], owner [5 distinct]):
50 pages, 200 tuples

Humans: (hid [int], age [1-100]):
1,000 pages, 50,000 tuples

Indexes:

1. B+ tree (unclustered) on Kitties.cuteness [5 pages]
2. B+ tree (unclustered) on Puppies.yappiness [5 pages]
3. B+ tree (clustered) on (Puppies.owner, Puppies.cuteness) [15 pages]
4. B+ tree (unclustered) on Humans.hid [20 pages]

Query Optimization Exercises

Tables:

Kitties: kid[int], cuteness [1-10], owner [10 distinct]):

100 pages, 400 tuples

Puppies: (pid [int], yappiness [1-10], owner [5 distinct]):

50 pages, 200 tuples

Humans: (hid [int], age [1-100]):

1,000 pages, 50,000 tuples

Indexes:

1. B+ tree (unclustered) on Kitties.cuteness [5 pages]
2. B+ tree (unclustered) on Puppies.yappiness [5 pages]
3. B+ tree (clustered) on (Puppies.owner, Puppies.cuteness) [15 pages]
4. B+ tree (unclustered) on Humans.hid [20 pages]

```
SELECT * FROM Kitties K, Puppies P, Humans H
WHEREK.owner = P.owner AND P.owner = H.hid
    AND P.yappiness = K.cuteness
  AND H.hid < 1200 AND P.yappiness = 7;
```

What are the best single table plans?

Query Optimization Exercises

```
SELECT * FROM Kitties K, Puppies P, Humans H
WHERE K.owner = P.owner
AND P.owner = H.hid
AND P.yappiness = K.cuteness
AND H.hid < 1200
AND P.yappiness = 7;
```

What are the best single table plans?

`Kitties` — file scan; no B+Trees.

`Puppies` — B+Tree on ‘yappiness’.

`Humans` — file scan: it faster than unclustered B+Tree!

Query Optimization Exercises

What are the 2-way joins that the optimizer will consider, and which ones will it throw out?

Kitties [file scan] ⋈ Puppies

Puppies [B+] ⋈ Kitties

Puppies [B+] ⋈ Humans

Humans [file scan] ⋈ Puppies

~~Humans [file scan] ⋈ Kitties~~

~~Kitties [file scan] ⋈ Humans~~

Query Optimization Exercises

Now let's consider the following query:

```
SELECT * FROM Kitties K, Puppies P
WHERE K.owner = P.owner
AND P.yappiness = K.cuteness
AND P.yappiness = 7;
```

What is the cost of doing page-oriented nested loops join with `Puppies` as the outer table?

Query Optimization Exercises

Now let's consider the following query:

```
SELECT * FROM Kitties K, Puppies P
WHERE K.owner = P.owner
AND P.yappiness = K.cuteness
AND P.yappiness = 7;
```

```
Puppies' Index Scan: 21 IOs (from #1)
5 * 100 IOs = 500 IOs
Total: 521 IOs
```

Query Optimization Exercises

Now let's consider the following query:

```
SELECT * FROM Kitties K, Puppies P
WHERE K.owner = P.owner
AND P.yappiness = K.cuteness
AND P.yappiness = 7;
```

What is the cost of doing page-oriented nested loops join with `Kitties` as the outer table?

Query Optimization Exercises

Now let's consider the following query:

```
SELECT * FROM Kitties K, Puppies P
WHERE K.owner = P.owner
AND P.yappiness = K.cuteness
AND P.yappiness = 7;
```

```
Kitties' File Scan: 100 IOs (from #1)
Puppies Index Lookup on (owner, yappiness):
(15 + 50 )*1/10*1/10 ~= 1
100 + 400 * 1 = 501 IOs
```