

Query Optimization

From Chapter 15

Schema for Examples

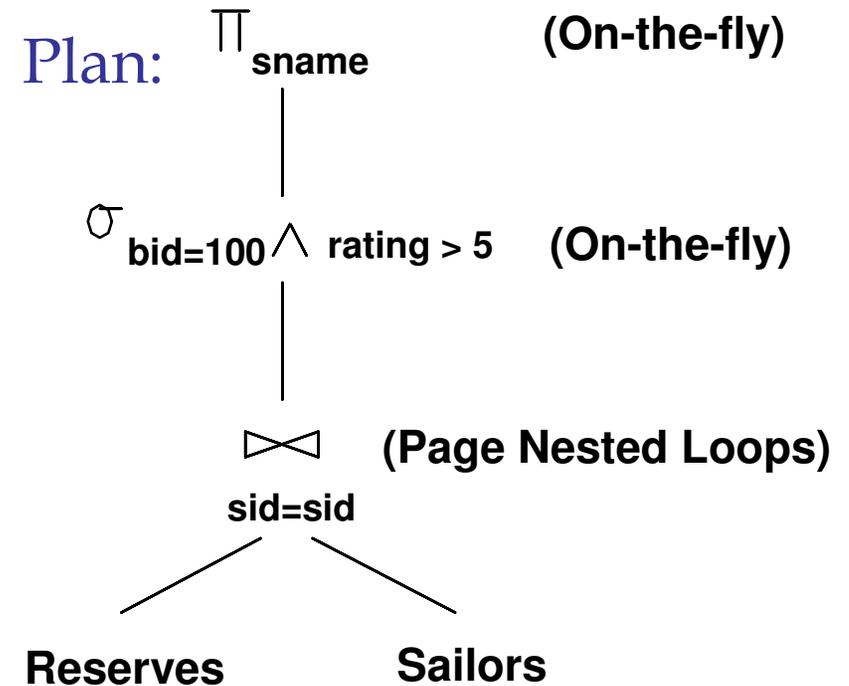
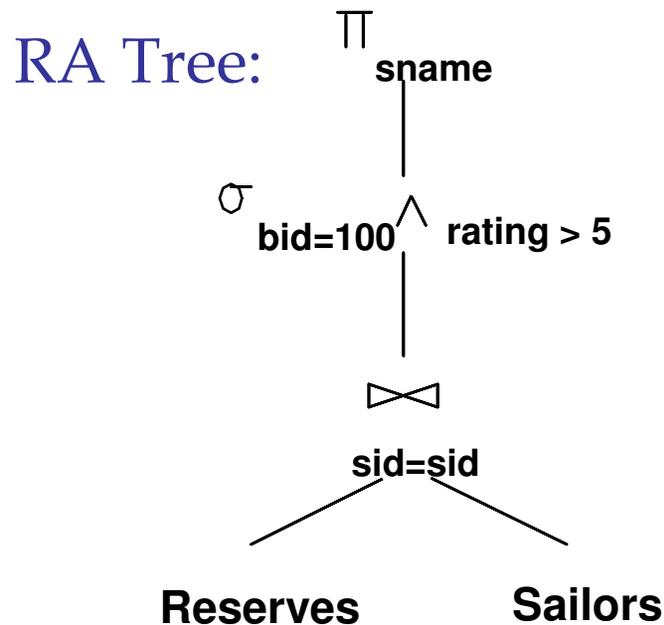
Sailors (sid: integer, sname: string, rating: integer, age: real)

Reserves (sid: integer, bid: integer, day: dates, rname: string)

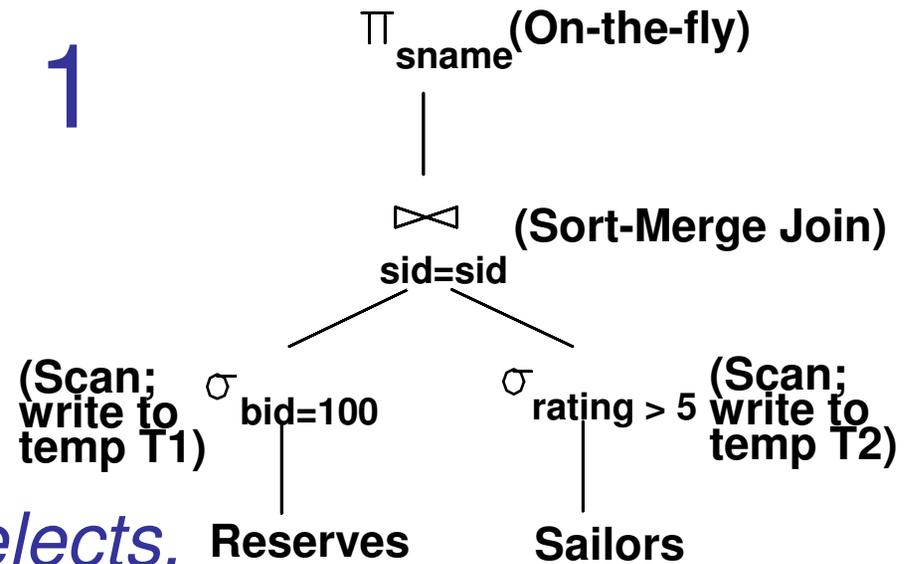
- Reserves:
 - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
 - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.

Motivating Example

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



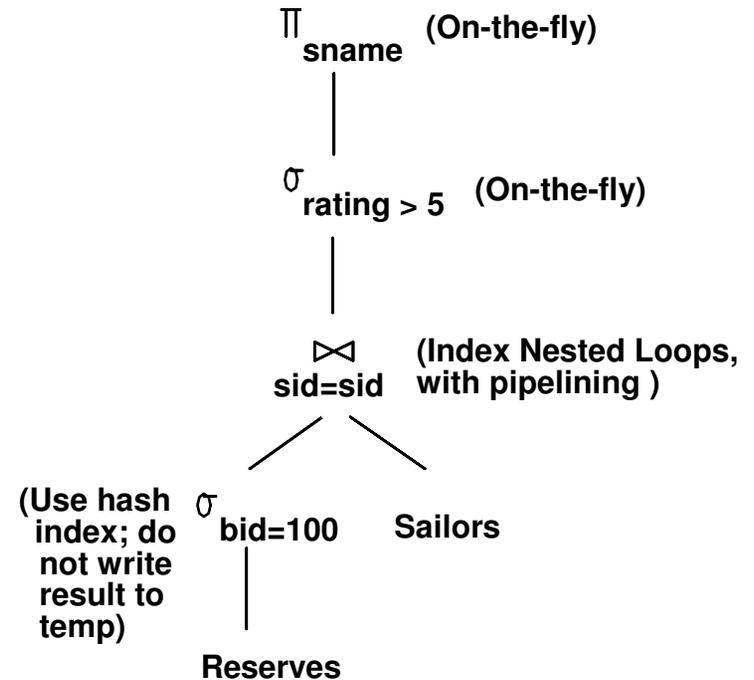
Alternative Plans 1 (No Indexes)



- **Main difference: push selects.**
- With 5 buffers, **cost of plan** (assume 100 boats, 10 ratings):
 - If we used BNL join, join cost = _____, **total cost** = _____.
 - 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.**

Alternative Plans 2 With Indexes

- 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).
 - ❖ Join column *sid* is a key for Sailors.
 - At most one matching tuple, unclustered index on *sid* _____
 - ❖ 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**.



Overview of Query Optimization

- Plan: *Tree of R.A. ops, with choice of alg for each op.*
 - Each operator typically implemented using a `pull` interface
- Two main issues:
 - For a given query, what plans are considered?
 - How is the cost of a plan estimated?

Outline

- Relational algebra equivalences
- Statistics and size estimation
- Plan enumeration and cost estimation
- Nested queries

Relational Algebra Equivalences

- Allow us to choose different join orders and to ‘push’ selections and projections ahead of joins.
 - Selections: $\sigma_{c_1 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\dots \sigma_{c_n}(R))$ (Cascade)
 $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$ (Commute)
 - ❖ Projections: $\pi_{a_1}(R) \equiv \pi_{a_1}(\dots (\pi_{a_n}(R)))$ (Cascade)
 - ❖ Joins: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (Associative)
 $(R \bowtie S) \equiv (S \bowtie R)$ (Commute)
- ☐ Show that: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

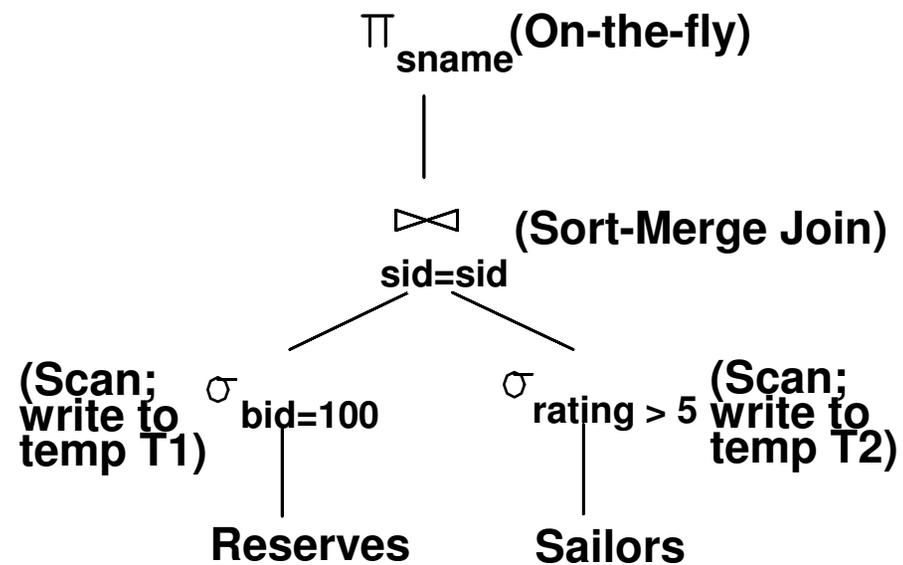
More Equivalences

- 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 $R \bowtie S$, we can 'push' it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

Outline

- Relational algebra equivalences
- **Statistics and size estimation**
- Plan enumeration and cost estimation
- Nested queries

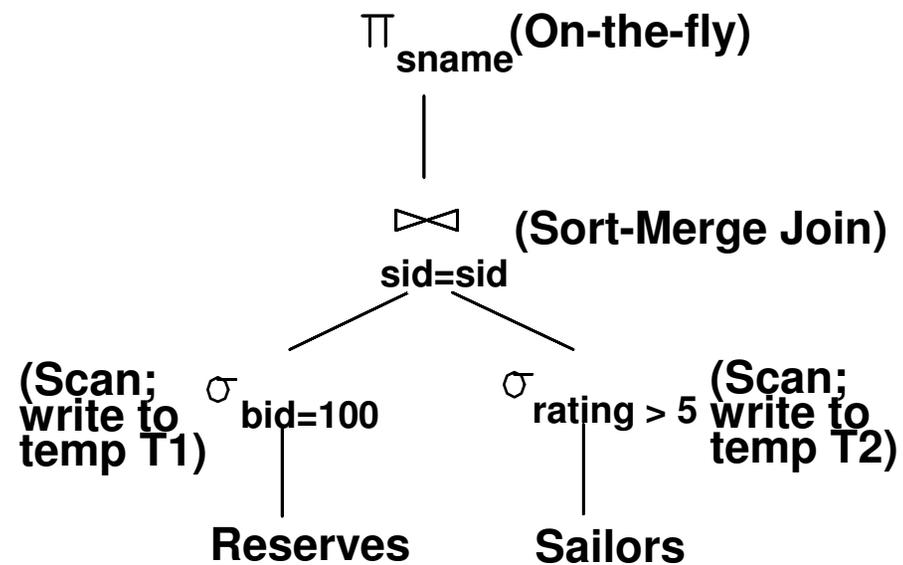
Example Plan



Statistics and Catalogs

- Need information about the relations and indexes involved. *Catalogs* typically contain at least:
 - Catalogs updated periodically.
 - More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Example Plan



Size Estimation and Reduction Factors

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

- Consider a query block:
- What is maximum # tuples possible in result?
- *Reduction factor (RF)* associated with each *term*
- *Result cardinality* = Max # tuples * product of all RF's.
 - 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 $(High(I)-value)/(High(I)-Low(I))$

Reduction Factors & Histograms

- For better estimation, use a histogram

No. of Values	2	3	3	1	8	2	1
Value	0-.99	1-1.99	2-2.99	3-3.99	4-4.99	5-5.99	6-6.99

equiwidth

No. of Values	2	3	3	3	3	2	4
Value	0-.99	1-1.99	2-2.99	3-4.05	4.06-4.67	4.68-4.99	5-6.99

equidepth

Outline

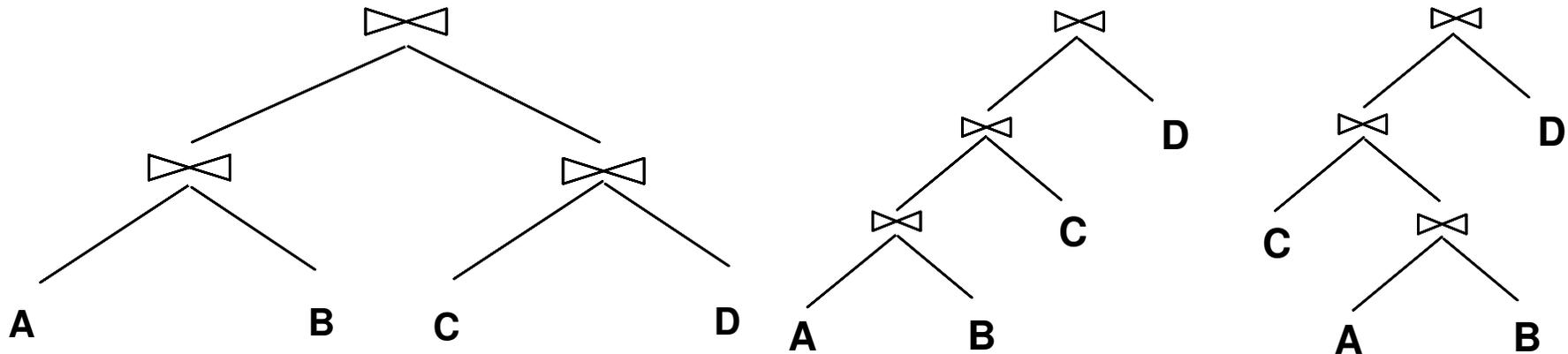
- Relational algebra equivalences
- Statistics and size estimation
- **Plan enumeration and cost estimation**
- Nested queries

Enumeration of Alternative Plans

- Two main cases:
- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:

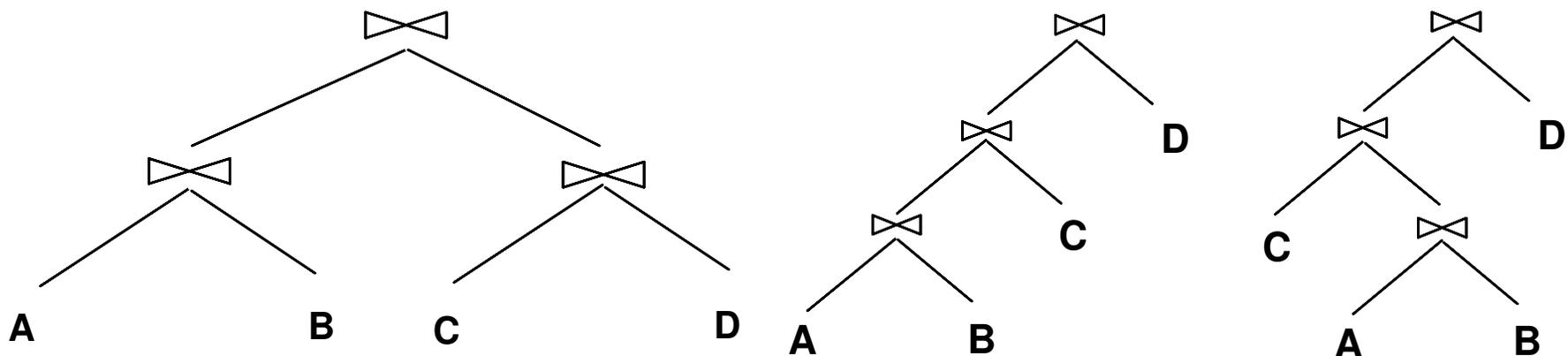
Queries Over Multiple Relations

- Fundamental decision in System R: only left-deep join trees are considered.



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*.



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:
 - Pass 2:
 - Pass N:
- For each subset of relations, retain only:
 - Cheapest plan overall
 - Cheapest plan for each *interesting order* of the tuples.

Example

Sailors:

B+ tree on *rating*

Hash on *sid*

Reserves:

B+ tree on *bid*

■ Pass 1:

- *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.

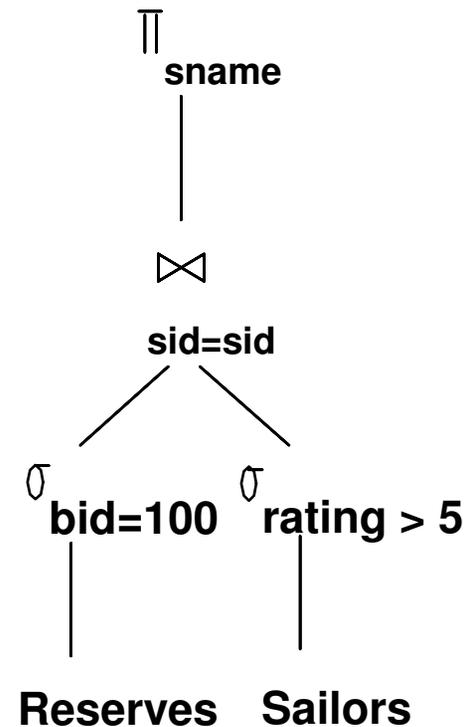
- Still, B+ tree plan kept (because tuples are in *rating* order).

- *Reserves*: B+ tree on *bid* matches *bid* = 100; 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.



Enumeration of Plans (Contd.)

- N-1 way plan not combined with a relation unless there is a join condition between them

- ORDER BY, GROUP BY, aggregates etc. handled as a final step
 - Use an 'interestingly ordered' plan
 - Or use an additional sorting operator

Example

Sailors:

Hash, B+ on *sid*

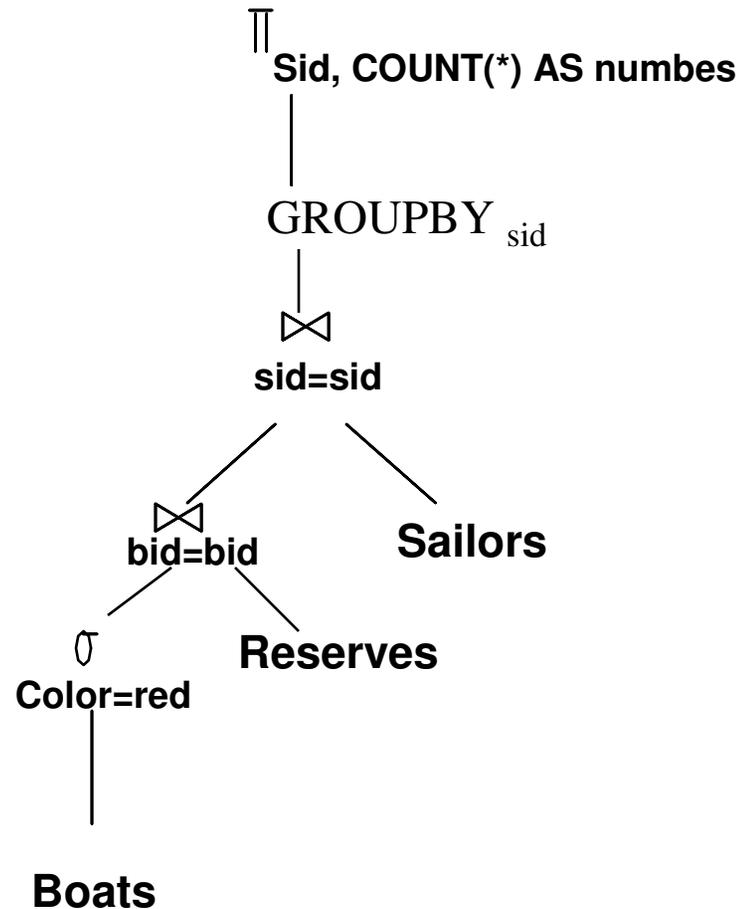
Reserves:

Clustered B+ tree on *bid*

B+ on *sid*

Boats

B+, Hash on *color*



Select S.sid, COUNT(*) AS numbes

FROM Sailors S, Reserves R, Boats B

WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = "red"

GROUP BY S.sid

Pass 1

- Best plan for accessing each relation regarded as the first relation in an execution plan

Pass 3

- For each of the plans retained from Pass 2, taken as the outer, generate plans for the inner join

Add cost of aggregate

- Cost to sort the result by sid, if not returned sorted

Outline

- Relational algebra equivalences
- Statistics and size estimation
- Plan enumeration and cost estimation
- **Nested queries**

Nested Queries

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of `calling` nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered.
The non-nested version of the query is typically optimized better.

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

Nested block to optimize:

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

Equivalent non-nested query: