CS 589, Principles of Database Systems, Lecture 3-2

Transformation-Based Optimization

Basis for a series of optimizer frameworks: Exodus, Volcano, Cascades, Columbia, Colombia

Optimize an expression: look for all equivalent plans

- requires optimizing sub-expressions
- "memoize" results for later reuse

Produce logically equivalent expressions via rewrite

(then map them to physical plans and estimate their costs)

Based on Algebraic Equivalences

Example:

$\sigma_P(u \cap v) \equiv \sigma_P(u) \cap \sigma_P(v)$

Need to know such equivalences are correct

Usually easiest to work from set-theoretic definition

$\sigma_P(r) = \{t \mid t \in r \text{ and } P(t)\}$

$r \cap s = \{t \mid t \in r \text{ and } t \in s\}$

David Maier & Lois Delcambre

Expand Left Side

$\sigma_P(u \cap v) =$

def. $\cap$

$\sigma_P(\{t1 \mid t1 \in u \text{ and } t1 \in v\}) =$

$\{t2 \mid t2 \in \{t1 \mid t1 \in u \text{ and } t1 \in v\} \text{ and } P(t2)\} =$

def of σ

$\{t2 \mid (t2 \in u \text{ and } t2 \in v) \text{ and } P(t2)\}$

logic

Expand Right Side

$\sigma_P(u) \cap \sigma_P(v) =$

Def σ

$\{t1 \mid t1 \in u \text{ and } P(t1)\} \cap \{t2 \mid t2 \in v \text{ and } P(t2)\} =$

Def $\cap$

$\{t3 \mid t3 \in \{t1 \mid t1 \in u \text{ and } P(t1)\} \text{ and } t3 \in \{t2 \mid t2 \in v \text{ and } P(t2)\}\} =$

$\{t3 \mid (t3 \in u \text{ and } P(t3)) \text{ and } (t3 \in v \text{ and } P(t3))\} =$

David Maier & Lois Delcambre

Continuing ...

{t3 | (t3 ∈ u and P(t3)) and
(t3 ∈ v and P(t3))} = [simplify]

{t3 | t3 ∈ u and P(t3) and t3 ∈ v} =
[rearrange]

{t3 | (t3 ∈ u and t3 ∈ v) and P(t3)} =
[change variable]

{t2 | (t2 ∈ u and t2 ∈ v) and P(t2)}

et voila!

Equivalences Can Have Side Conditions

An equivalence may hold only under certain conditions

$\pi_X(\sigma_P(r)) \equiv \sigma_P(\pi_X(r))$
if mentioned(P) $\subseteq$ X

mentioned(P) = set of attributes used in P

mentioned = AB

$\pi_{ABC}(\sigma_{A>B}(r)) \equiv \sigma_{A>B}(\pi_{ABC}(r))$ OK

$\pi_{BC}(\sigma_{A>B}(r)) \equiv \sigma_{A>B}(\pi_{BC}(r))$ not OK

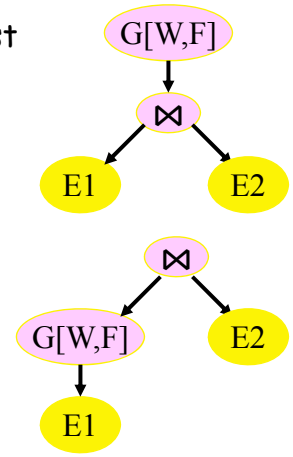
not well formed.

David Maier & Lois Delcambre

Rewriting Group-by

Initial translation usually puts group-by above join

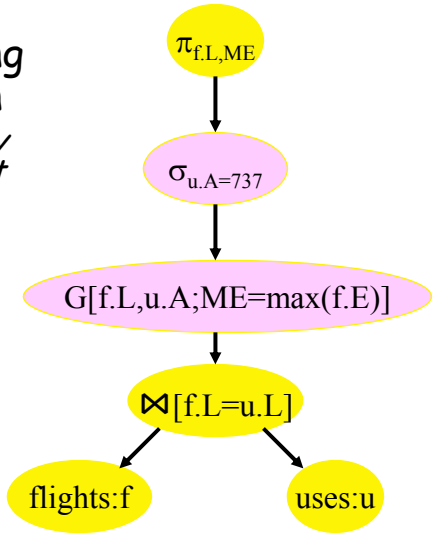
- May be useful to move past other operators
- Move down: makes one of the join inputs smaller
- Move up: might make input to group-by smaller



Select and Group-by

Consider the following starting expression

Find max # of economy seats for flights that use a 737

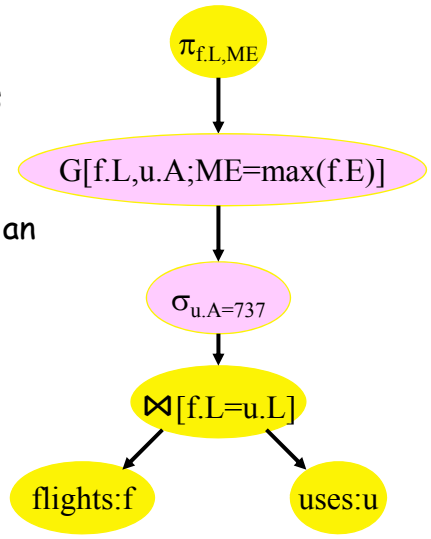


David Maier & Lois Delcambre

Do Selection Early

Is this okay?

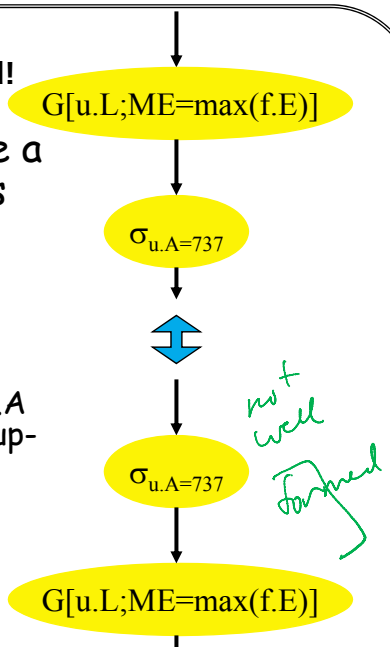
- Yes – the select applies to the grouping attributes, which are constant for a group
- Want groups to pass in an all-or-nothing manner



I'm Confused!

Paper* says you can move a select if the attributes it uses are functionally determined by the grouping attributes

- Assume L → A on uses
- Doesn't seem right to me: u.A isn't visible above the group-by



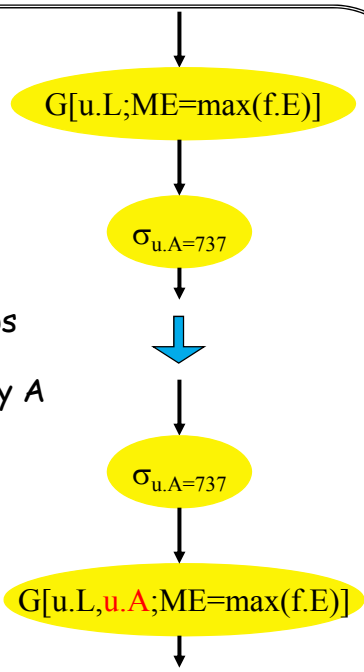
* *Orthogonal Optimization of Subqueries and Aggregation*

David Maier & Lois Delcambre

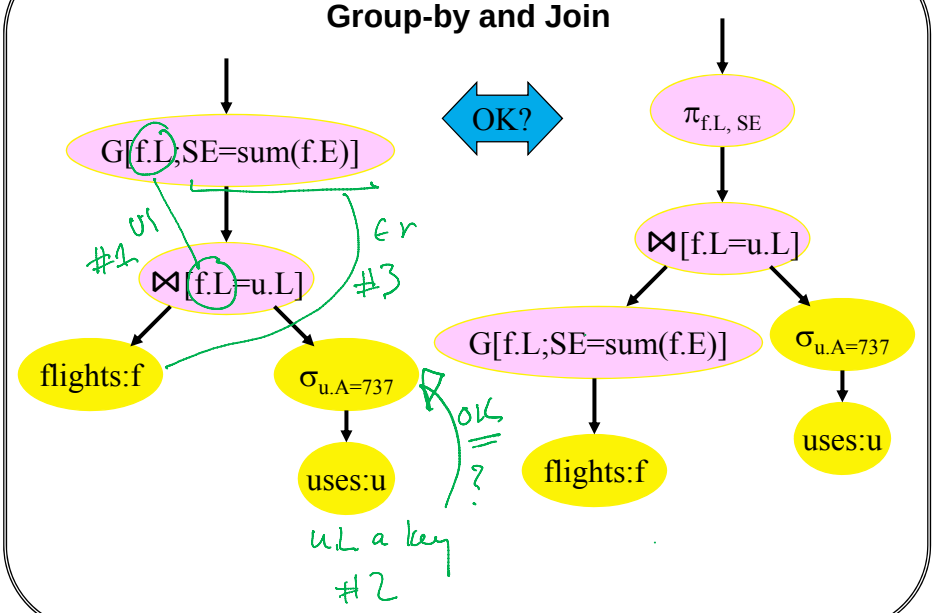
Add an Attribute?

If we lift a selection above a Group-by, add the attribute to the grouping condition?

- Won't create more groups if L → A.
- Will need to project away A at some point.



Group-by and Join



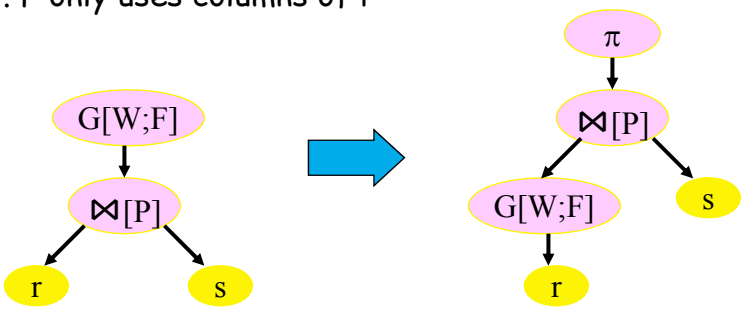
David Maier & Lois Delcambre

Side Conditions

Have r(R), s(S), P mentions $X \subseteq R$ (join condition)

Group-by through join okay if

1. X is in W: join columns of R included in grouping columns
2. W contains a key of s
3. F only uses columns of r



Why?

1. Can't lose the join columns
2. Need to have only one s tuple per row of G[W;F]
   - Can relax for max and min
3. Only have r available

Transforming from right to left, conditions 1 and 3 come for free

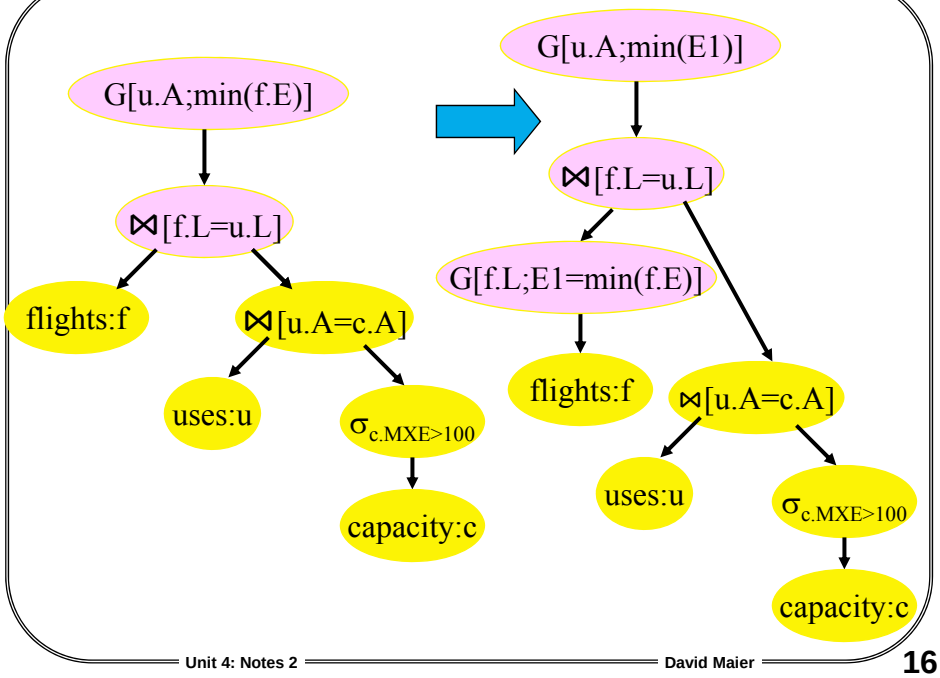
- Given that the initial expression is well formed

David Maier & Lois Delcambre

Splitting a Group-by

Can do partial grouping (local group-by), then a join, then rest of grouping

```
select u.A, min(f.E)
from flight as f, uses as u,
  capacity as c
where f.L = u.L and u.A = c.A
  and c.MXE > 100
group-by u.A
```

David Maier & Lois Delcambre

Min of Mins = Min of all

u.A	f.L	f.E
737	115	65
737	115	70
737	115	75
737	80	95
737	80	75

u.A	f.L	E1
737	115	65
737	80	75

u.A	min(E1)
737	65

Works for Sum, Too

u.A	f.L	f.E
737	115	65
737	115	70
737	115	75
737	80	95
737	80	75

u.A	f.L	E1
737	115	210
737	80	170

u.A	sum(E1)
737	380

E1 = sum(f.E)

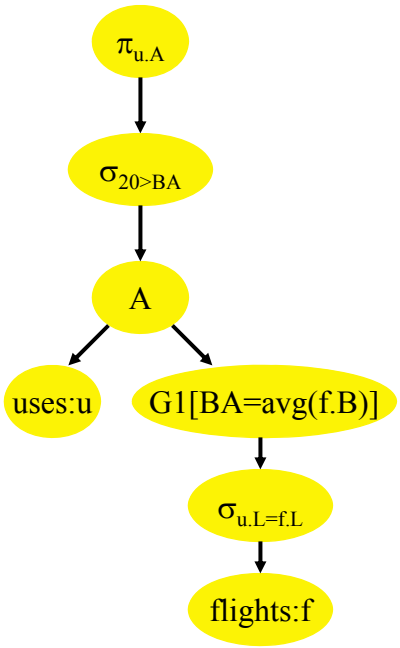
What about count? Does count of counts = count of all?
What about average??

David Maier & Lois Delcambre

Removing Sub-queries

We introduced "apply" to translate subqueries

```
select u.A
from uses as u
where 20 >    [BA]
  (select avg(f.B)  [as BA]
   from flights as f
   where f.L = u.L)
```

David Maier & Lois Delcambre

Apply Options

Can have a direct (for-loop) implementation of apply as a physical operator

But it is also possible to remove apply via rewriting

Strategy:

- Use push-down transforms that move apply towards bottom of expression tree
- Until its right input no longer depends on the left input
- Then, convert apply to a join

Example: Apply through Scalar Group-by

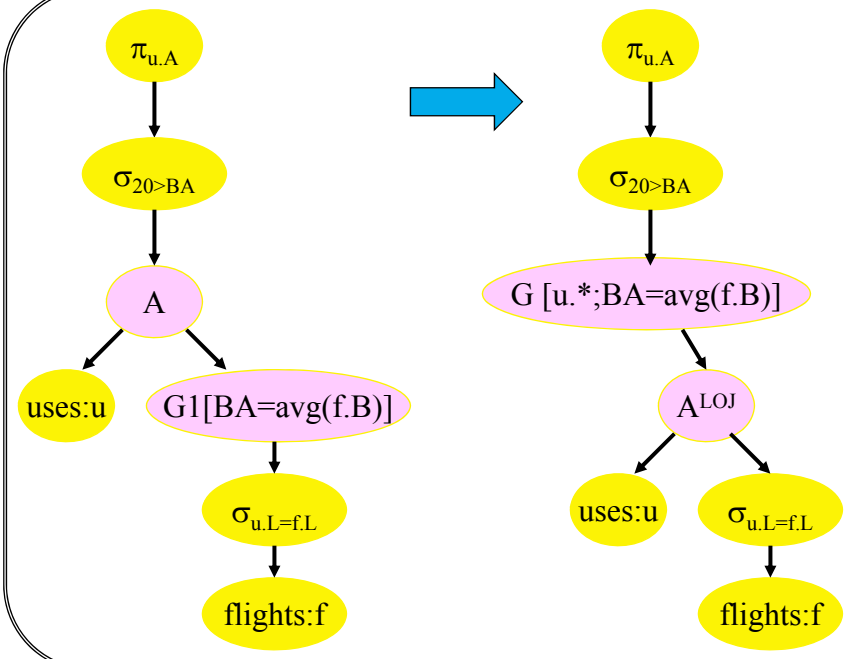
$r\ A\ G1[F](e(x)) \rightarrow G[r.*,F](r\ A^{LOJ}\ e(x))$

A^{LOJ} keeps every tuple t in r, even if e(t) is empty

- need it because G1 always produces a row

```
r A^LOJ e(x) =
r LOJ (DE(r) A e(x))
```


David Maier & Lois Delcambre



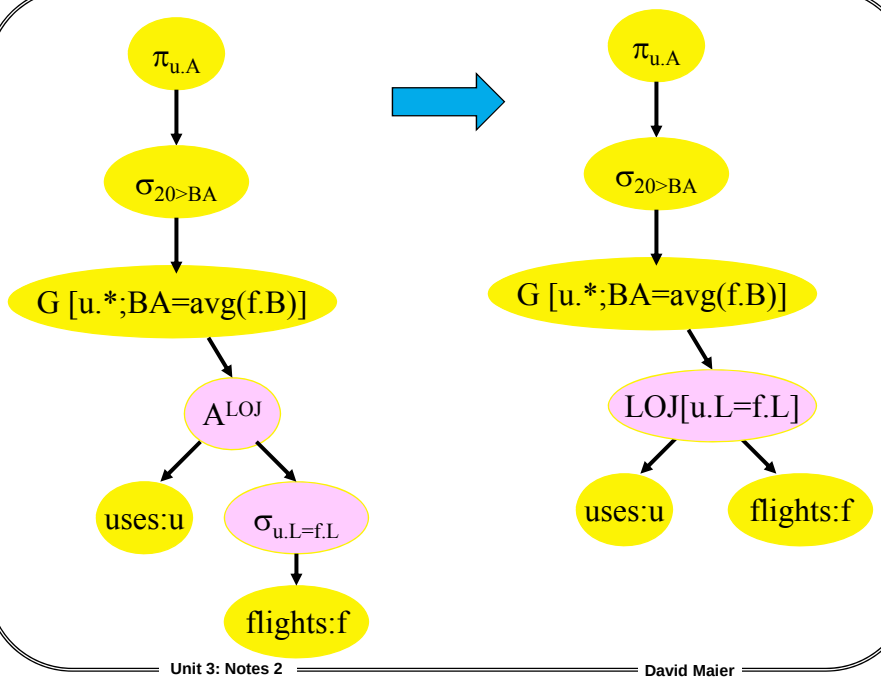
Example: Apply-to-Join Conversion

$r\ A^{LOJ}\ \sigma_P(e) \rightarrow r\ \ LOJ[P]\ e$ ~ not under no mention of r

if e does not depend on r
(okay for P to depend on r)

Can think of apply as a "catalyst": used in transforming algebra, but doesn't show up in the end product

David Maier & Lois Delcambre



What Was I Lying About?

Not all query plans are created equal

They differ in *physical properties*

- sort order
- distribution
- compression

Some physical operators need inputs with certain properties in order to work correctly

Sort-merge join: needs inputs sorted on attributes in an equality join condition

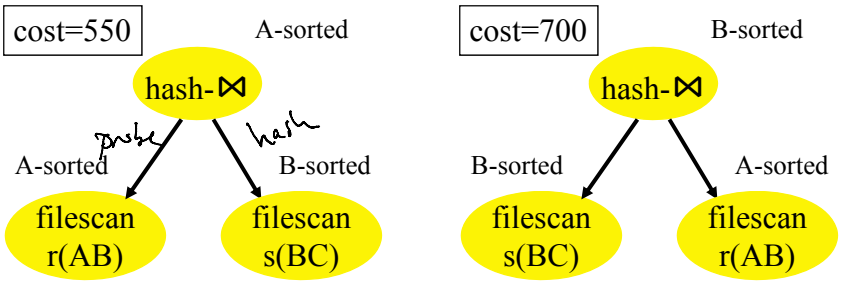
David Maier & Lois Delcambre

Physical Properties Can Affect Cost

Union, project might be faster on compressed inputs
- Fewer bytes to deal with

Physical Properties and Optimization

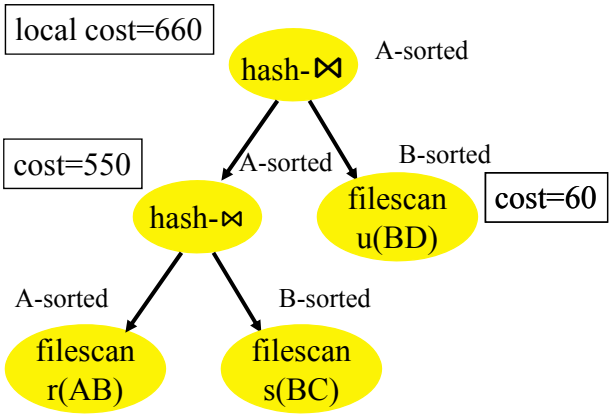
Because of differing physical properties, cheapest sub-plan might not be best choice in overall plan.
- consider join of r, s and u

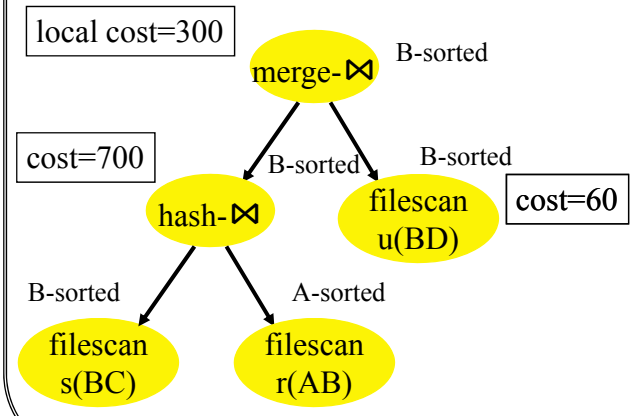


David Maier & Lois Delcambre

Now Join with u(BD)

The more expensive sub-plan might allow a cheaper join with u(BD)





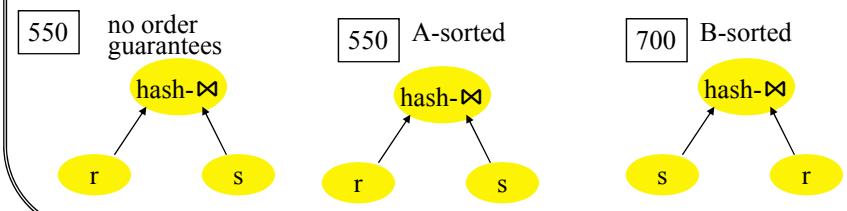
David Maier & Lois Delcambre

So, Do We Forget the Optimality Principle?

No – it still holds, relative to plans with the same physical properties.

So – need to keep best plan for each physical property

For example, dynamic programming approach keeps the best plan for each collection of physical properties



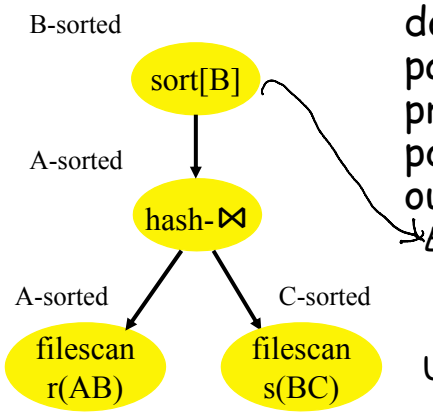
Do We Need Plans for Every Possible Property?

Probably not. For example, System R only keeps best plans for each *interesting order* (sort orders that might be useful in rest of query)

- attributes in a join predicate
- grouping attributes in an aggregate
- order-by clause in original query

David Maier & Lois Delcambre

Can Enforce Properties



Even if a sub-plan doesn't produce a particular physical property, usually possible to modify output to do so.

Enforcer: physical operator that enforces a physical property, but is a "no-op" at the logical level

Usually incurs some cost

Mapping Logical to Physical

Not a one-to-one correspondence generally between logical and physical operators

- Can have several physical ops that correspond to a given logical op

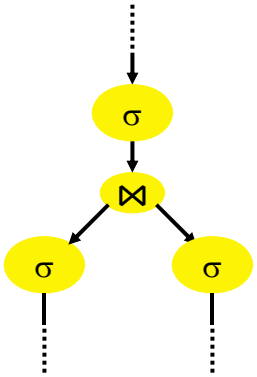
```
join: nested-loops join, index-
  nested-loops join, hash join,
  merge join
```

- One physical op might implement several logical ops

```
Have flags on hash join for regular
  join, outer-join, semi-join
```


David Maier & Lois Delcambre

- One physical op might translate a tree of logical ops



hash-join(join-pred, pre-left, pre-right, post-pred)

Cost Estimation

Can view operators under alternative interpretations

Normal view
- $\sigma_{A>B}$: relation → relation

Alternative interpretations

Properties
- $\sigma_{A>B}$: order → order, FDs → FDs

Stats
- $\sigma_{A>B}$: column histogram → column histogram

Costs
- $\sigma_{A>B}$: CPU secs → CPU secs

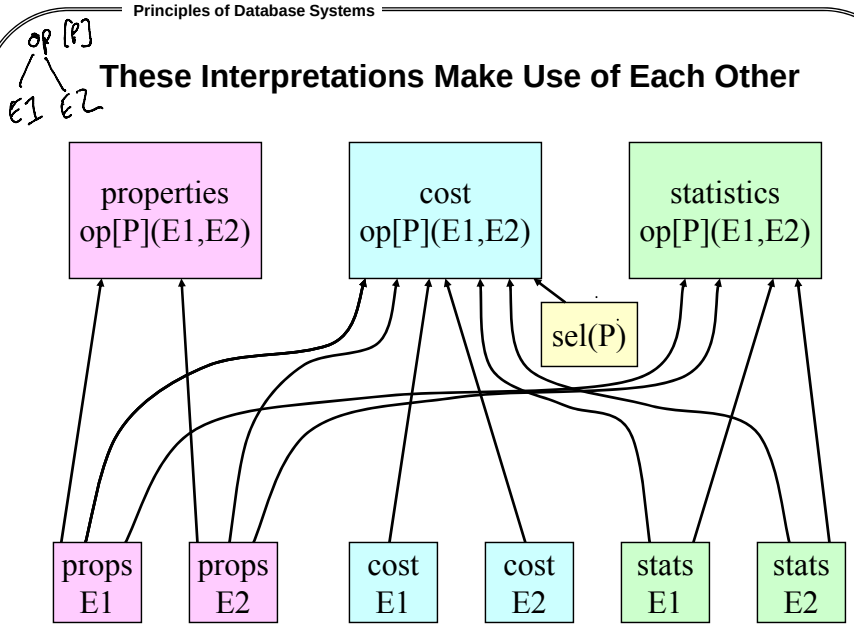
David Maier & Lois Delcambre

Also for Selection and Join Predicates

Interpret predicates as a *selectivity*:
% of tuples (or tuple pairs) expected to satisfy the condition

System R
- sel(A=5) = 10%
- sel(A<5) = 50%

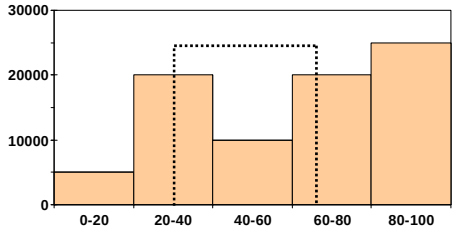
These Interpretations Make Use of Each Other



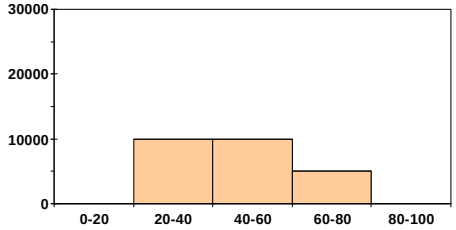
David Maier & Lois Delcambre

Histogram Example (on Attribute A)

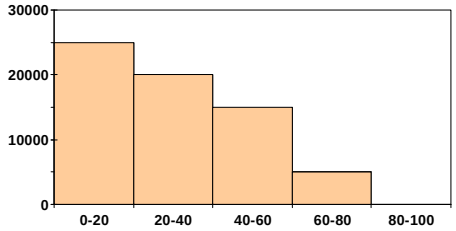
r(AB)



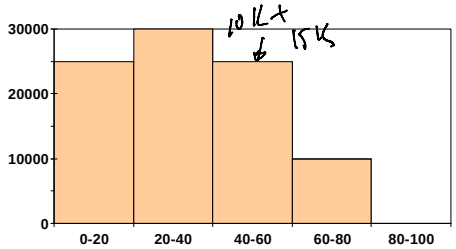
$\sigma_{30 \le A \le 65}(r(AB))$



s(AB)

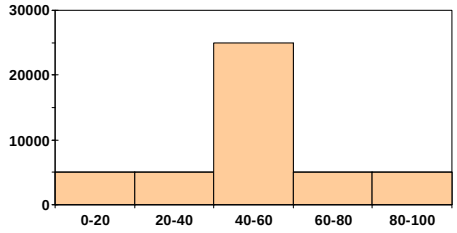


$\sigma_{30 \le A \le 65}(r(AB)) \cup s$

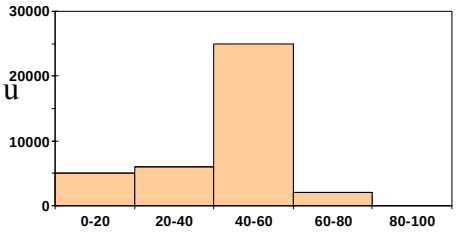


David Maier & Lois Delcambre

u(AC)



$(\sigma_{30 \le A \le 65}(r(AB)) \cup s) \bowtie u$



David Maier & Lois Delcambre