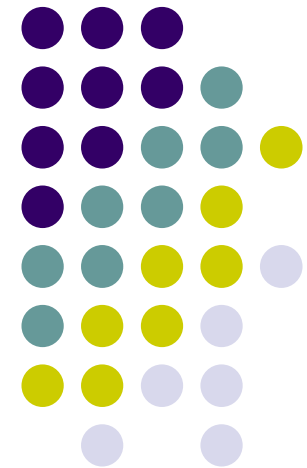
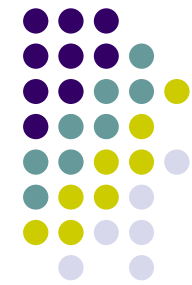


Relational Algebra and Query Optimization

Chapter 4, 15

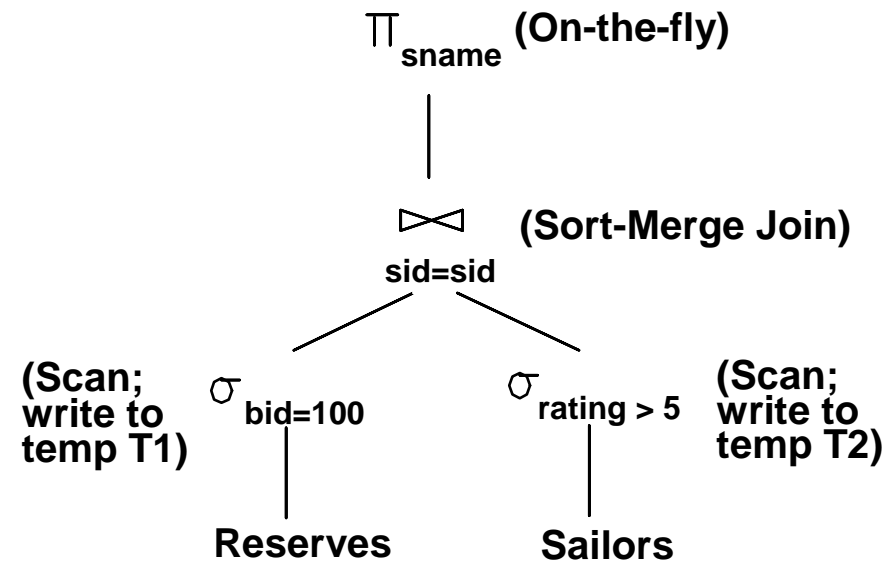
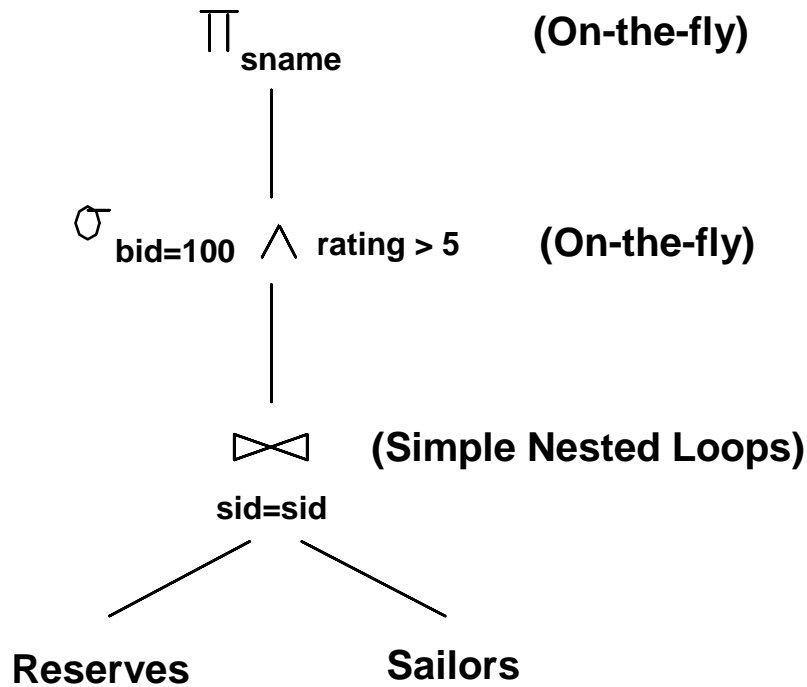


Goal: Optimizing execution plan using RA-tree representations

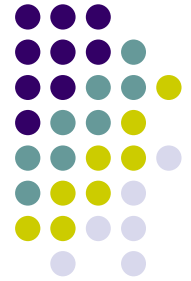


```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
      R.bid=100 AND S.rating>5
```

- Execution plans for query
- Alternative RA-trees produce same result but differ in computational efficiency



Relational Query Languages



- Query languages:
Allow manipulation and **retrieval of data** from a database.
- Relational model supports simple, powerful QLs:
 - Strong formal foundation based on logic.
 - Allows for much optimization.
- Query Languages **!=** programming languages!
 - QLs not expected to be “Turing complete”.
 - QLs not intended to be used for complex calculations.
 - QLs support easy, efficient access to large data sets.

Preliminaries



- A query is applied to *relation instances*, and the result of a query is also a relation instance.
 - *Schemas of input* relations for a query are *fixed* (but query will run regardless of instance!)
 - The *schema for the result* of a given query is also *fixed!* Determined by definition of query language constructs.

Example Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation, assume that names of fields in query results are ‘inherited’ from names of fields in query input relations.

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

S1

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra



- Basic operations:
 - Selection (σ) Selects a subset of rows from 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 and in reln. 2.
- Additional operations:
 - Intersection, join, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be composed

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates!* (Why??)
- Duplicate elimination can be costly:
O(M²), M=number of pages,
if no index is defined on relation.
Why?

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname, rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Selection



- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)
- Equality selection can be supported by tree ($O(\ln M)$) or hash index ($O(1)$).
- Inequality supported by B+ tree index (see Ch. 8)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating > 8} (S_2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating} (\sigma_{rating > 8} (S_2))$$

Union, Intersection, Set-Difference



- All of these operations take two input relations, which must be union-compatible:
 - Same number of fields.
 - `Corresponding' fields have the same type.
- What is the schema of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$



Cross-Product (\times)

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
 - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- Renaming operator: $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

Joins



- Condition Join: $R \bowtie_c S = \sigma_c (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Joins (not Joints!)



- Equi-Join: A special case of condition join where the condition c contains only **equalities**.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

- Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- Natural Join: Equijoin on *all* common fields.

$$S1 \bowtie R1$$

Processing a Join

$$S \bowtie R$$



- Join processing can make database access slow!
- No Index: Time = $O(M n)$,
M: #pages S, n: #tuples R
- No Index on S, Clustered B+ Tree on R
Time = $O(M \log n)$
- Clustered B+ Tree index on common attributes:
Time = $O(M+N)$,
M: #pages S, N: #pages R
- Processing time can be optimized by choosing good processing sequence

Find names of sailors who've reserved boat #103



Solution 1: $\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

Solution 2: $\rho(Temp1, \sigma_{bid=103} Reserves)$

$\rho(Temp2, Temp1 \bowtie Sailors)$

$\pi_{sname}(Temp2)$

Solution 3: $\pi_{sname}(\sigma_{bid=103}(Reserves \bowtie Sailors))$



Find sailors who've reserved a red or a green boat

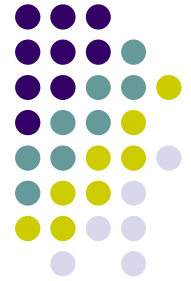
- First identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho (Tempboats, (\sigma_{color='red' \vee color='green'} Boats))$$
$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

We can also define Tempboats using union! (How?)

What happens if \vee is replaced by \wedge in this query?

Find sailors who've reserved a red and a green boat



- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Relational Algebra Equivalences



- Allow us to choose different join orders and to `push` selections and projections ahead of joins → execution time minimization

- Selections: $\sigma_{c_1 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\dots \sigma_{c_n}(R))$ (*Cascade*)

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \quad (\text{Commute})$$

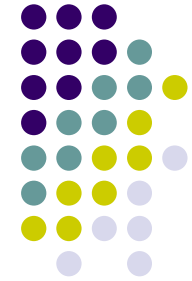
- ❖ Projections: $\pi_{a_1}(R) \equiv \pi_{a_1}(\dots(\pi_{a_n}(R)))$ (*Cascade*)

- ❖ Joins: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ (*Associative*)

$$(R \bowtie S) \equiv (S \bowtie R) \quad (\text{Commute})$$

+ Show that: $R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S$

More Equivalences/ Pushing Selection



- 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.
- Pushing selections can reduce cost significantly.

Find names of sailors who've reserved a red boat



- Information about boat color only available in Boats; so need an extra join:

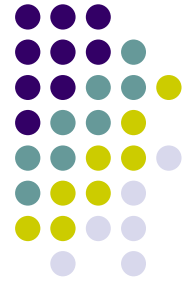
$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

A more efficient solution (selections early):

$$\pi_{sname}(\pi_{sid}((\pi_{bid} \sigma_{color='red'} Boats) \bowtie Res) \bowtie Sailors)$$

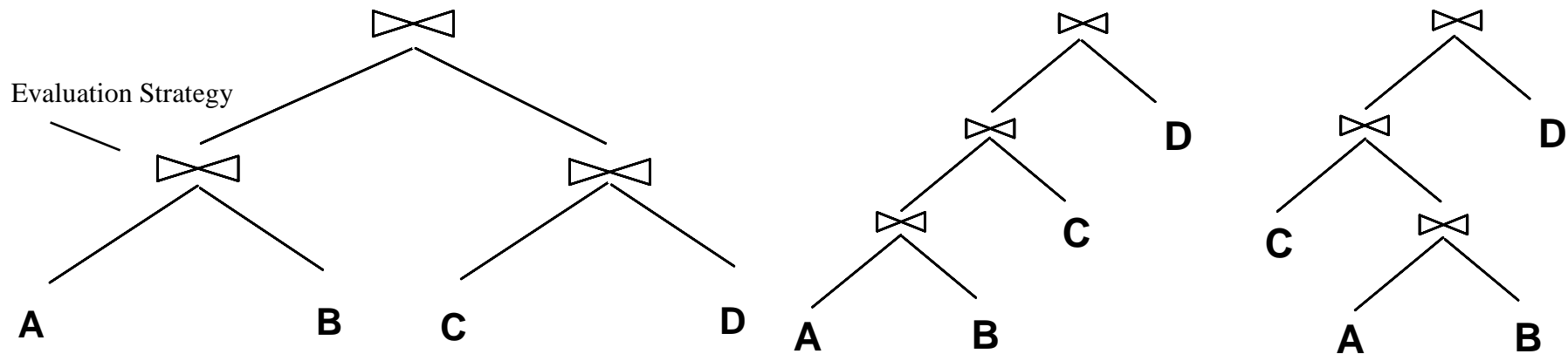
A query optimizer can find this, given the first solution!

Relational Query Optimization



- RA Queries can be represented as trees
- Query optimizes RA tree
- Typical Question: How can we reduce time for processing joins?
 - How can indexes be set to support joins?
 - How can we do projection and selection as early as possible?
 - How can we find the optimal sequence of joins exploiting associativity?
- The DBMS translates the query into RA tree and finds best processing strategy based on heuristics and DB catalog data (e.g. size of tables)

Plan space: Different possibilities to evaluate joins:

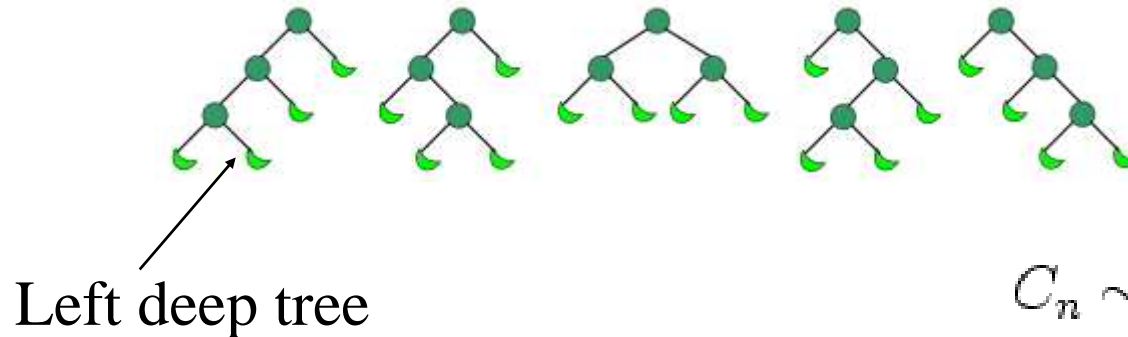


- First level: Space of all sorted, rooted trees -> Joins (associativity)
- Second level: Labels in sorted trees can be permuted -> commutative law
- Third level: Different possibilities to handle single selections, joins

The space of ordered, rooted trees



- Structurally different Ordered, rooted trees with 4 leaves (3 joins)



- General: n : Number of relations, $n-1$: Number of joins, C_n : Number of join trees (Catalan numbers)
- Number grows very fast: Therefore, often only left deep trees considered by optimizer (can be processed 'on the fly')
- Number of left deep trees is still grows exponentially with number of tables, but enumeration feasible for small number of joins (<10)

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:** Find best 1-relation plan for each relation.
 - **Pass 2:** Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
 - **Pass N:** Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
 - Cheapest plan overall, plus
 - Cheapest plan for each *interesting order* of the tuples.

Summary



- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.