Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#15: Query Optimization

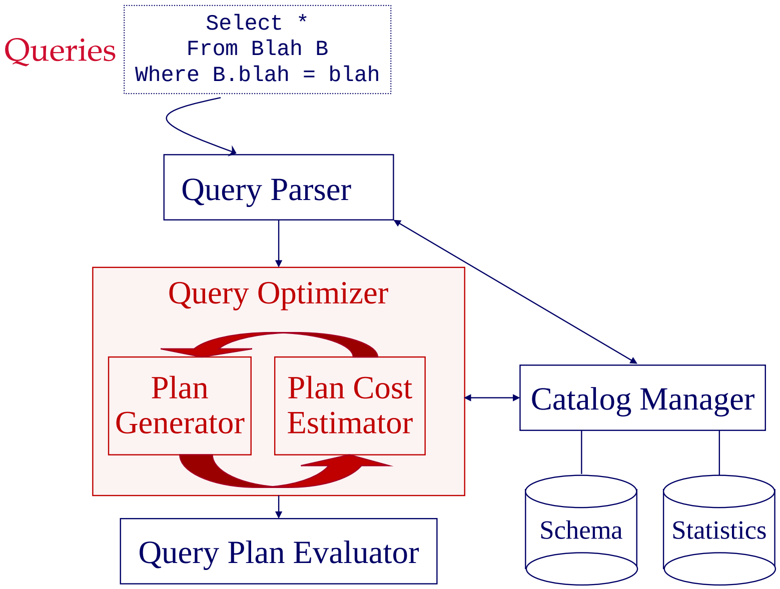
Administrivia

- HW5 is due Thursday March 17th.
- You can pick up your mid-term from Marilyn Walgora’s office (GHC 8120).
- Christos is out of the country.
- No office hours next week.

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration

Cost-based Query Sub-System



```
Select *
From Blah B
Where B.blah = blah
```

Query Optimization

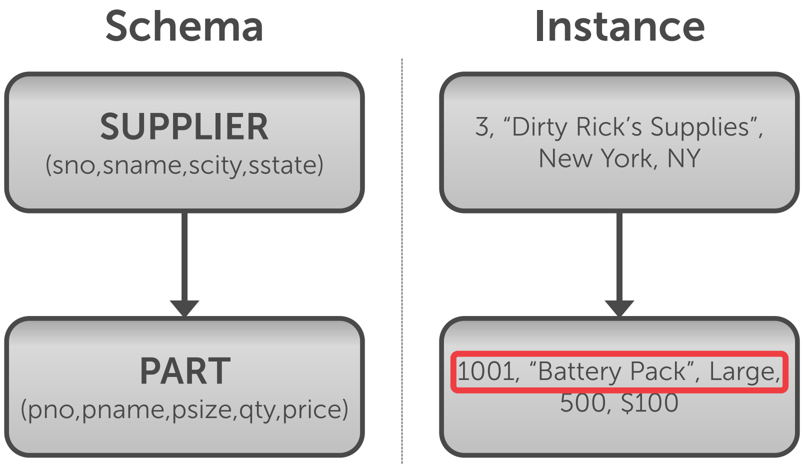
- Remember that SQL is declarative.
  - User tells the DBMS *what* answer they want, not *how* to get the answer.
- There can be a big difference in performance based on plan is used:
  - See last week: 5.7 days vs. 45 seconds

Quick DB History Lesson

1960s – IBM IMS

- First database system.
- Hierarchical data model.
- Programmer-defined physical storage format.
- Tuple-at-a-time queries.

Hierarchical Data Model



Schema

SUPPLIER
(sno,sname,scity,sstate)

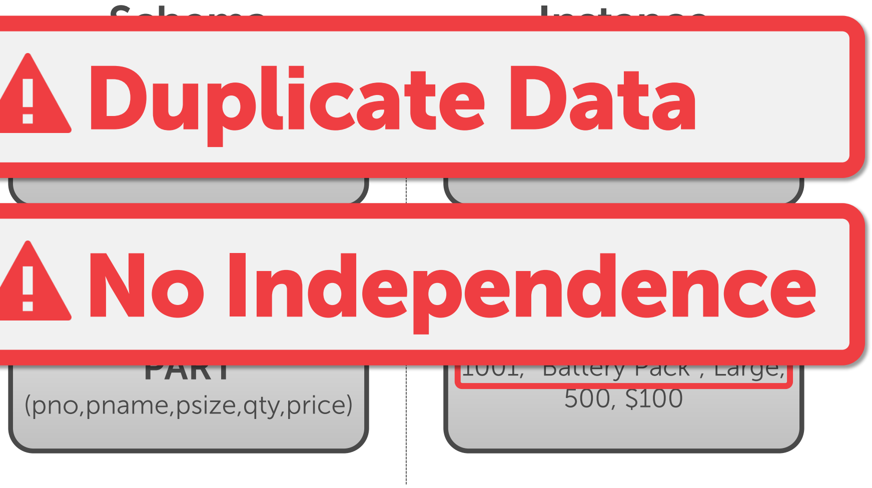
PART
(pno,pname,psize,qty,price)

Instance

3, “Dirty Rick’s Supplies”, New York, NY

1001, “Battery Pack”, Large, 500, $100

Hierarchical Data Model



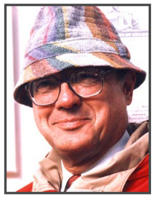
Schema

Instance

⚠ Duplicate Data

⚠ No Independence

1970s – CODASYL

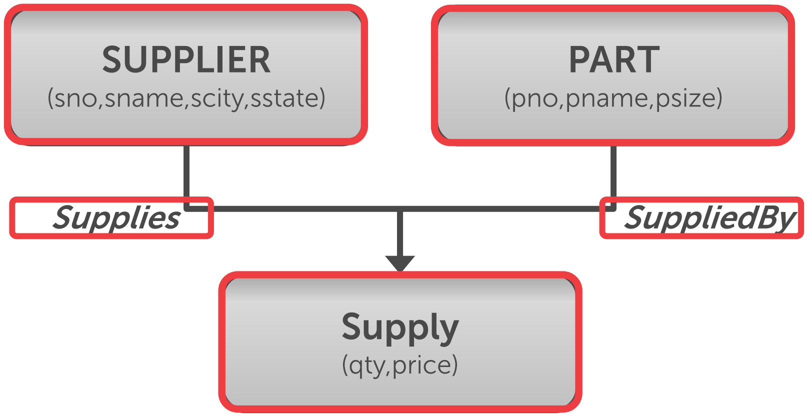


Bachman

- COBOL people got together and proposed a standard based on a network data model.
- Tuple-at-a-time queries.
  - – This forces the programmer to do manual query optimization.

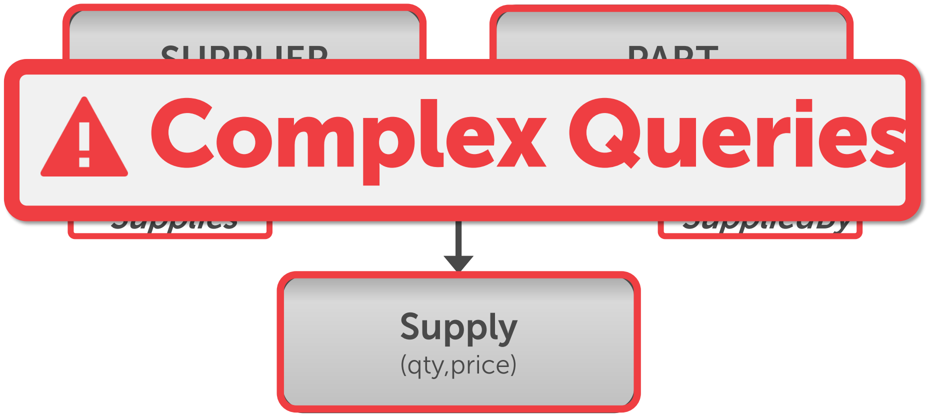
Network Data Model

Schema



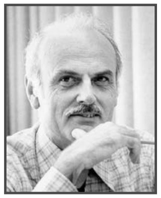
Network Data Model

Schema



⚠ Complex Queries

1970s – Relational Model


Codd

- Ted Codd saw the maintenance overhead for IMS/Codasyl.
- Proposed database abstraction based on relations:
  - Store database in simple data structures.
  - Access it through high-level language.
  - Physical storage left up to implementation.

IBM System R

- Skunkworks project at IBM Research in San Jose to implement Codd’s ideas.
- Had to figure out all of the things that we are discussing in this course themselves.
- IBM never commercialized System R.

IBM System R

- First implementation of a query optimizer.
- People argued that the DBMS could never choose a query plan better than what a human could write.
- A lot of the concepts from System R’s optimizer are still used today.

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration
- Nested Sub-queries

Relational Algebra Equivalences

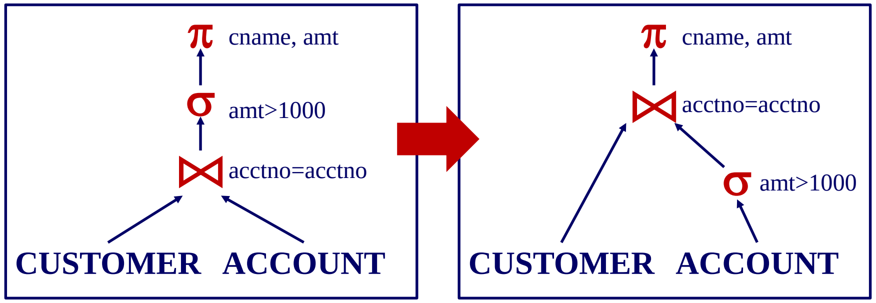
- A query can be expressed in different ways.
- The optimizer considers variations and choose the one with the lowest cost.
- How do we know whether two queries are equivalent?
  - Equivalence Rules (chapter 15.3)

Relational Algebra Equivalences

- Two relational algebra expressions are ***equivalent*** if they generate the same set of tuples.

Relational Algebra Equivalences

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```



Relational Algebra Equivalences

```
SELECT cname, amt
  FROM customer, account
 WHERE customer.acctno =
       account.acctno
   AND account.amt > 1000
```

$$\pi_{\text{cname, amt}}(\sigma_{\text{amt}>1000}(\text{customer} \bowtie \text{account}))$$

$$=$$

$$\pi_{\text{cname, amt}}(\text{customer} \bowtie (\sigma_{\text{amt}>1000}(\text{account})))$$

Equivalence of Expressions

- Q: How to prove a transf. rule?

$$\sigma_P(R1 \bowtie R2) = \sigma_P(R1) \bowtie \sigma_P(R2)$$

- Use relational tuple calculus to show that LHS = RHS:

$$\underbrace{\sigma_P(R1 \cup R2)}_{\textbf{LHS}} = \underbrace{\sigma_P(R1) \cup \sigma_P(R2)}_{\textbf{RHS}}$$

Equivalence of Expressions

$$\sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2)$$

$$\Downarrow$$

$$t \in LHS \Leftrightarrow$$

$$t \in (R1 \cup R2) \wedge P(t) \quad \Leftrightarrow$$

$$(t \in R1 \vee t \in R2) \wedge P(t) \quad \Leftrightarrow$$

$$(t \in R1 \wedge P(t)) \vee (t \in R2) \wedge P(t)) \quad \Leftrightarrow$$

Equivalence of Expressions

$$\sigma_P(R1 \cup R2) = \sigma_P(R1) \cup \sigma_P(R2)$$

$$\Downarrow$$

…

$$(t \in R1 \wedge P(t)) \vee (t \in R2) \wedge P(t)) \quad \Leftrightarrow$$

$$(t \in \sigma_P(R1)) \vee (t \in \sigma_P(R2)) \quad \Leftrightarrow$$

$$t \in \sigma_P(R1) \cup \sigma_P(R2) \quad \Leftrightarrow$$

$$t \in RHS$$

$$QED$$

Equivalence of Expressions

- Q: How to disprove a rule?

R1

A	B
Christos	squirrels

R2

A	B
Christos	knifefights

$$\pi_A(R1 - R2) \cancel{=} \pi_A(R1) - \pi_A(R2)$$

$$\Downarrow \qquad\qquad \Downarrow$$

A	B
Christos	squirrels

$\neq$ Ø

Equivalence of Expressions

- **Selections:**
  - Perform them early
  - Break a complex predicate, and push

$$\sigma_{p1 \wedge p2 \wedge \ldots pn}(R) = \sigma_{p1}(\sigma_{p2}(\ldots\sigma_{pn}(R))\ldots)$$

- Simplify a complex predicate
  - (`X=Y AND Y=3`) → `X=3 AND Y=3`

Equivalence of Expressions

- **Projections:**
  - Perform them early
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - Project out all attributes except the ones requested or required (e.g., joining attr.)

Equivalence of Expressions

- **Joins:**
  - Commutative, associative

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

- Q: How many different orderings are there for an *n*-way join?

Equivalence of Expressions

- **Joins:** How many different orderings are there for an n-way join?
- A: Catalan number ~ 4^n
  - Exhaustive enumeration: too slow.
- We’ll see in a second how an optimizer limits the search space...

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration

Query Optimization

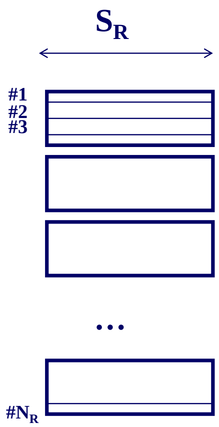
- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
- Estimate cost for each plan.
- Pick the best one.

Cost Estimation

- How long will a query take?
  - **CPU**: Small cost; tough to estimate.
  - **Disk**: # of block transfers.
  - **Network**: # of messages
- How many tuples will qualify?
- What statistics do we need to keep?

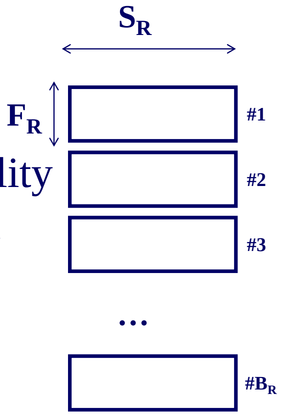
Cost Estimation – Statistics

- For each relation **R** we keep:
  - **N_R** → # tuples
  - **S_R** → size of tuple in bytes
  - **V(A,R)** → # of distinct values of attribute ‘A’



Derivable Statistics

- **F_R** → max# records/block
- **B_R** → # blocks
- **SC(A,R)** → selection cardinality avg# of records with A=given

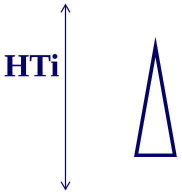


Derivable Statistics

- **SC(A,R)** → Selection Cardinality avg# of records with A=given → **N_R / V(A,R)**
- Note that this assumes data uniformity
  - 10,000 students, 10 colleges – how many students in SCS?

Additional Statistics

- For index **i**:
  - **F_i** → average fanout (~50-100)
  - **HT_i** → # levels of index **i** (~2-3) ~ log(#entries)/log(**F_i**)
  - **LB_i** # → blocks at leaf level



Statistics

- Where do we store them?
- How often do we update them?

Selection Statistics

- We saw simple predicates (`name=“Christos”`)
- How about more complex predicates, like
  - `salary > 10000`
  - `age=30 AND jobTitle=“Costermonger”`
- What is their selectivity?

Selections – Complex Predicates

- Selectivity **sel(P)** of predicate **P**:

  == fraction of tuples that qualify

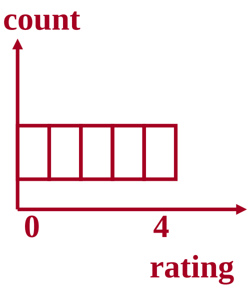
  sel(P) = **SC(P)** / N_R

\# of tuples

Selection Cardinality

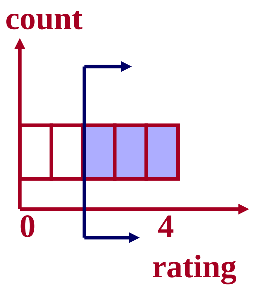
Selections – Complex Predicates

- Assume that **V(rating, SAILORS)** has 5 distinct values (i.e., 0 to 4).
- simple predicate **P**: A=constant
  - **sel(A=constant)** = **1/V(A,R)**
  - eg., **sel(rating=‘2’)** = 1/5
- What if **V(A,R)** is unknown??



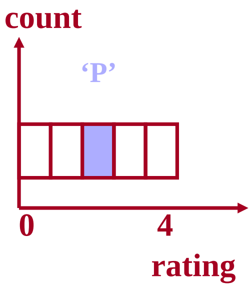
Selections – Complex Predicates

- Range Query: **sel(**rating >= ‘2’**)**
- **sel(**A>a**)** = $(A_{max} - a) / (A_{max} - A_{min})$



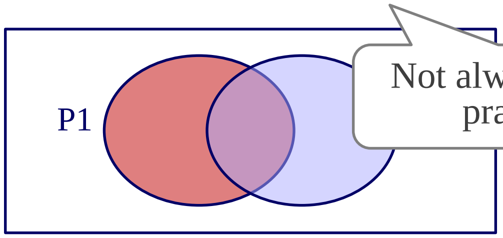
Selections – Complex Predicates

- Negation: **sel(**rating != ‘2’**)**
  - **sel(not** P) = **1 – sel(**P)
- Observation: selectivity ≈ probability



Selections – Complex Predicates

- **Conjunction:**
  - **sel(**rating = ‘2’ **and** name LIKE ‘C%’**)**
  - **sel(**P1 ∧ P2**) = sel(**P1**) · sel(**P2**)**
  - INDEPENDENCE ASSUMPTION



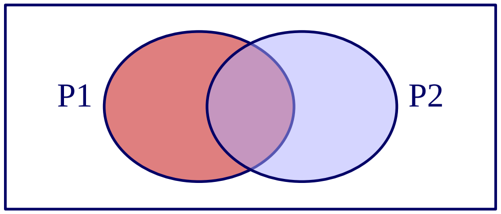
Not always true in practice!

Selections – Complex Predicates

- **Disjunction**:
  - **sel(**rating = ‘2’ **or** name LIKE ‘C%’**)**
  - **sel(**P1 ∨ P2)
    = **sel(**P1) + **sel(**P2) – **sel(**P1 ∨ P2)
    = **sel(**P1) + **sel(**P2) – **sel(**P1) · **sel(**P2)
  - INDEPENDENCE ASSUMPTION, again

Selections – Complex Predicates

- **Disjunction, in general**:
  - **sel(**P1 **or** P2 **or** … Pn) =
  - 1 - (1- **sel(**P1)) · (1 - **sel(**P2)) · … (1 - **sel(**Pn))



Selections – Summary

- sel(A=constant) → 1/V(A,r)
- sel(A>a) → $(A_{max} – a) / (A_{max} – A_{min})$
- sel(**not** P) → 1 – sel(P)
- sel(P1 **and** P2) → sel(P1) · sel(P2)
- sel(P1 **or** P2) → sel(P1) + sel(P2) – sel(P1) · sel(P2)
- sel(P1 **or** ... **or** Pn) = 1 - (1-sel(P1)) · ... · (1-sel(Pn))

Joins

- Q: Given a join of **R** and **S**, what is the range of possible result sizes in #of tuples?
  - Hint: what if $\mathbf{R}_{cols} \cap \mathbf{S}_{cols} = \emptyset$?
  - $\mathbf{R}_{cols} \cap \mathbf{S}_{cols}$ is a key for **R** and a foreign key in **S**?

Joins

- Q: Given a join of **R** and **S**, what is the range of possible result sizes in #of tuples?
  - Hint: what if $\mathbf{R}_{cols} \cap \mathbf{S}_{cols} = \emptyset$?
  - $\mathbf{R}_{cols} \cap \mathbf{S}_{cols}$ is a key for **R** and a foreign key in **S**?

$\mathbf{N_R \cdot N_S}$

$\leq \mathbf{N_S}$

Result Size Estimation for Joins

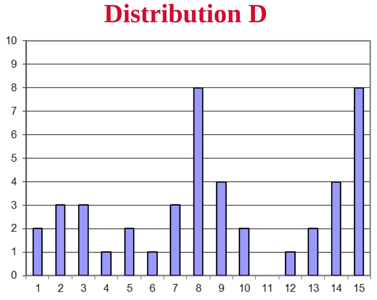
- General case: $\mathbf{R}_{cols} \cap \mathbf{S}_{cols}$ = {A} where A is not a key for either table.
- *Hint: for a given tuple of R, how many tuples of S will it match?*

Result Size Estimation for Joins

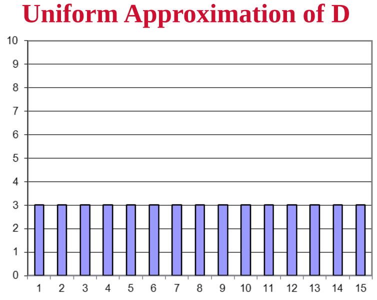
- General case: $\mathbf{R_{cols}} \cap \mathbf{S_{cols}} = \{A\}$ where A is not a key for either table.
  - Match each **R**-tuple with **S**-tuples:
    estSize ≈ $N_R \cdot N_S / V(A,S)$
  - Symmetrically, for **S**:
    estSize ≈ $N_R \cdot N_S / V(A,R)$
- Overall:
  - estSize ≈ $N_R \cdot N_S / \max(\{V(A,S), V(A,R)\})$

Cost Estimations

- Our formulas are nice but we assume that data values are uniformly distributed.



Distribution D

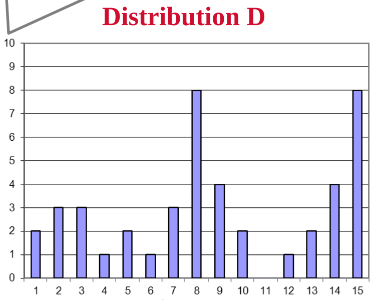


Uniform Approximation of D

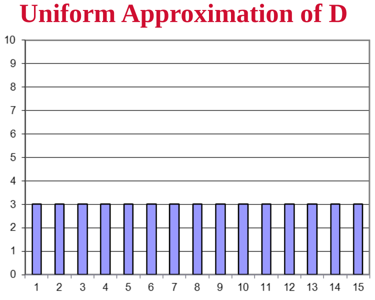
Cost Estimations

- Our formulas are nice but we assume that data values are uniformly distributed.

# of occurrences



Distribution D

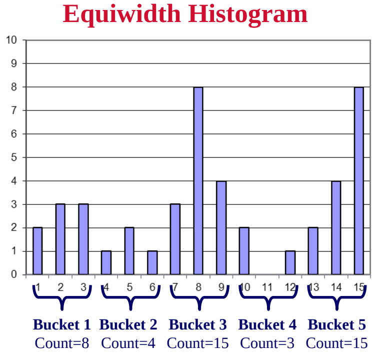


Uniform Approximation of D

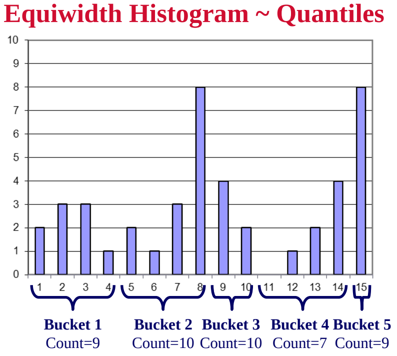
Distinct values of attribute

Histograms

- Allows the DBMS to have leverage better statistics about the data.



Equiwidth Histogram



Equiwidth Histogram ~ Quantiles

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration

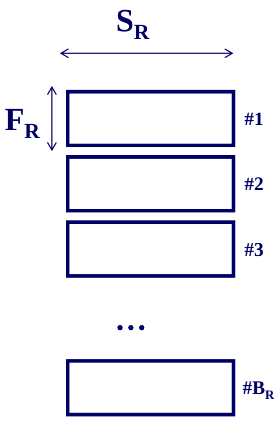
Query Optimization

- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - Multiple relations.
- Estimate cost for each plan.
- Pick the best one.

Plan Generation

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

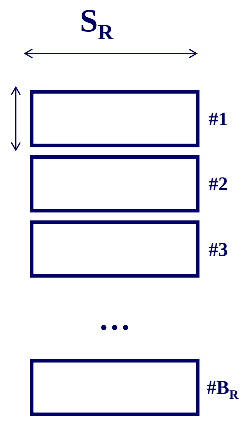
- What are our plan options?



Plan Generation

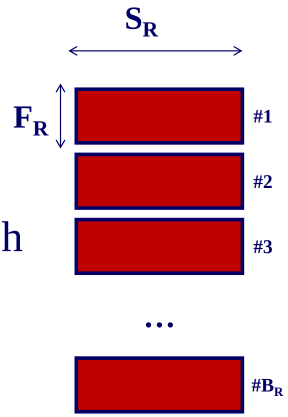
```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- Sequential Scan
- Binary Search
  - *if sorted & consecutive*
- Index Search
  - *if an index exists*



Sequential Scan

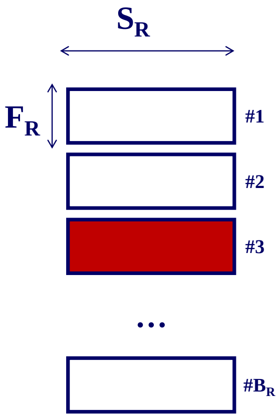
```sql
SELECT *
  FROM SAILORS
WHERE rating = 10
```



- **B_R** (worst case)
- **B_R /2** (on average, if we search for primary key)

Binary Search

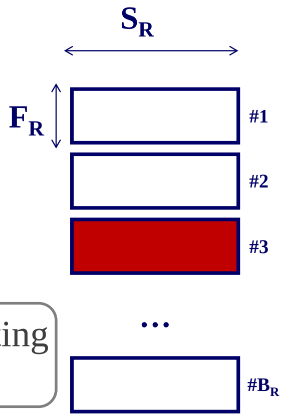
```sql
SELECT *
  FROM SAILORS
WHERE rating = 10
```



- **~log(B_R) + SC(A,R)/ F_R**
- Extra blocks are ones that contain qualifying tuples

Binary Search

```sql
SELECT *
  FROM SAILORS
WHERE rating = 10
```

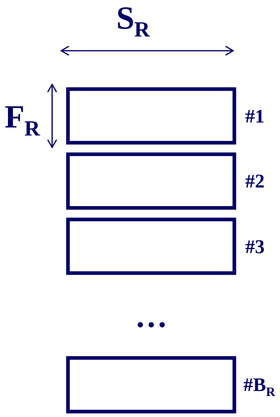


- **~log(B_R) + SC(A,R)/ F_R**
- Extra blocks a e ones that contain quali g tuples

We showed that estimating this is non-trivial.

Index Search

```sql
SELECT *
  FROM SAILORS
WHERE rating = 10
```



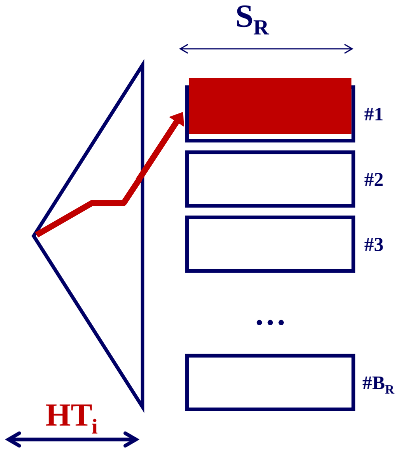
- Index Search:
  - levels of index + blocks w/ qual. tuples

Case#1: Primary Key
Case#2: Secondary key – clustering index
Case#3: Secondary key – non-clust. index

Index Search: Case #1

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

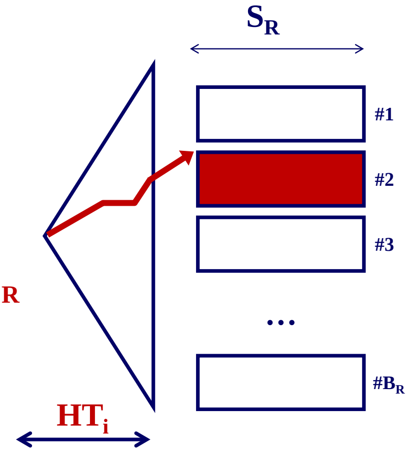
- Primary Key
  - cost: **$HT_i + 1$**



Index Search: Case #2

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

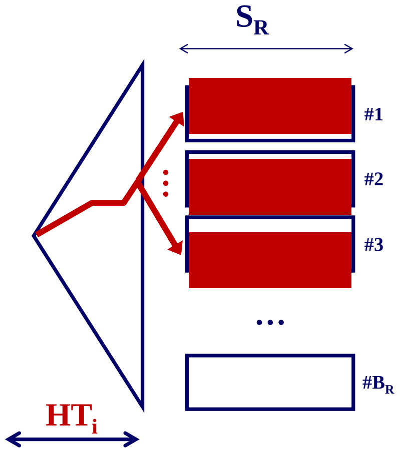
- Secondary key with clustering index:
  - cost: **$HT_i + SC(A,R)/F_R$**



Index Search: Case #3

```
SELECT *
  FROM SAILORS
WHERE rating = 10
```

- Secondary key with non-clustering index:
  - cost: **$HT_i + SC(A,R)$**



Query Optimization

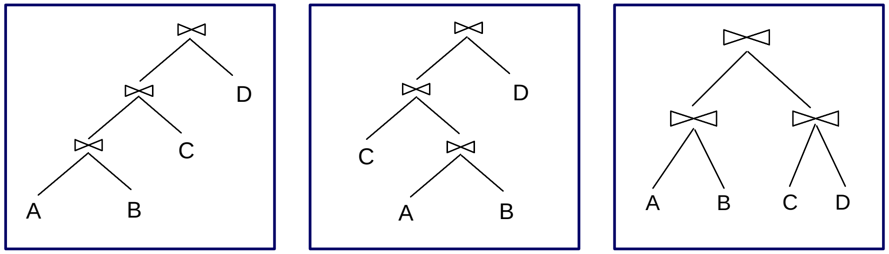
- Bring query in internal form (eg., parse tree)
- … into "canonical form" (syntactic q-opt)
- Generate alternative plans.
  - ➡ Single relation.
  - Multiple relations.
- Estimate cost for each plan.
- Pick the best one.

Queries over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly
  - We need to restrict search space.
- **Fundamental decision in System R:** only left-deep join trees are considered.

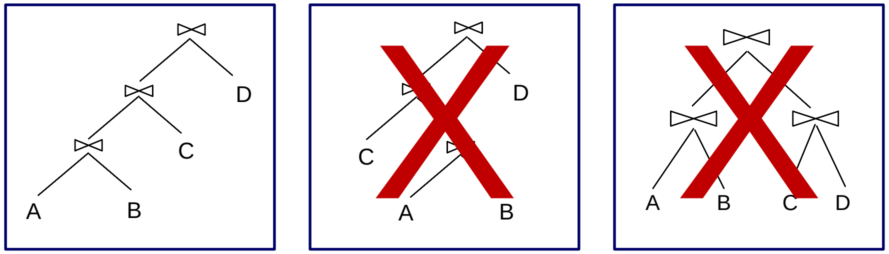
Queries over Multiple Relations

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



Queries over Multiple Relations

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



Queries over Multiple Relations

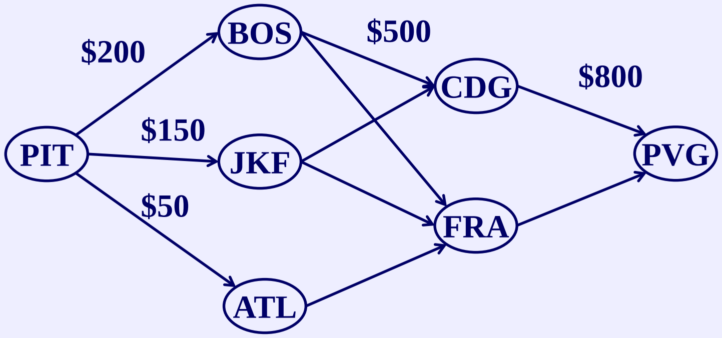
- **Fundamental decision in System R:** only left-deep join trees are considered.
  - Allows for fully pipelined plans where intermediate results not written to temp files.
  - Not all left-deep trees are fully pipelined (e.g., SM join).

Queries over Multiple Relations

- Enumerate the orderings (= left deep tree)
- Enumerate the plans for each operator
- Enumerate the access paths for each table

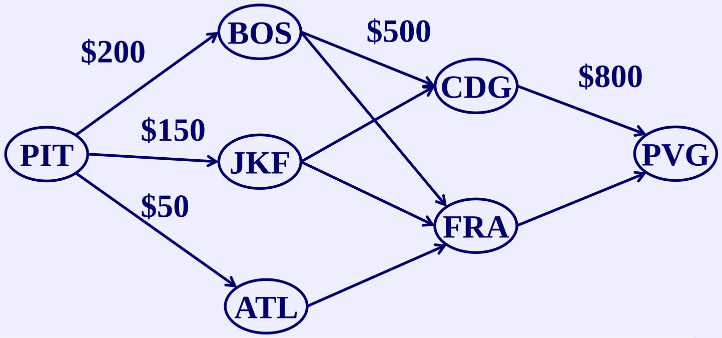
- Use **dynamic programming** to save cost estimations.

Dynamic Programming Example



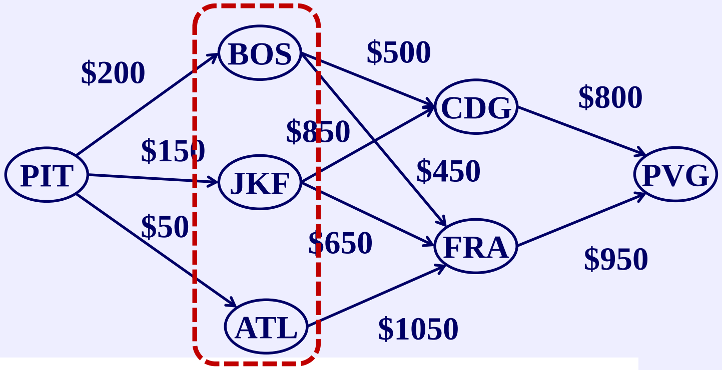
Cheapest flight PIT -> PVG?

Dynamic Programming Example



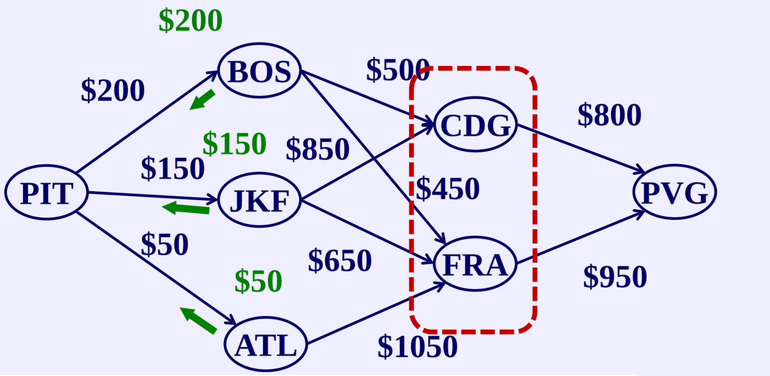
Assumption: NO package deals: cost CDG->PVG is always $800, no matter how reached CDG

Dynamic Programming Example



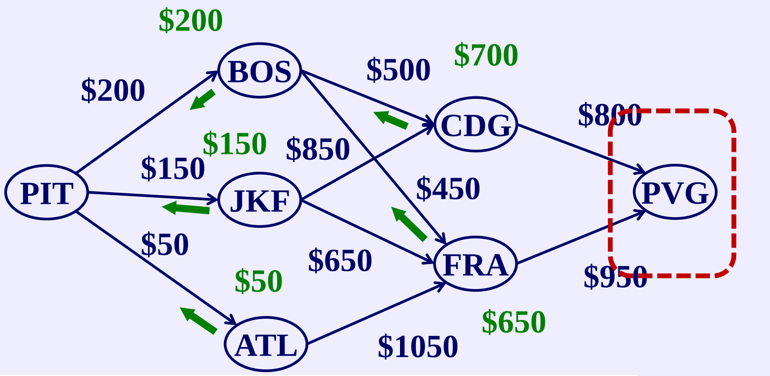
Solution: compute partial optimal, left-to-right:

Dynamic Programming Example



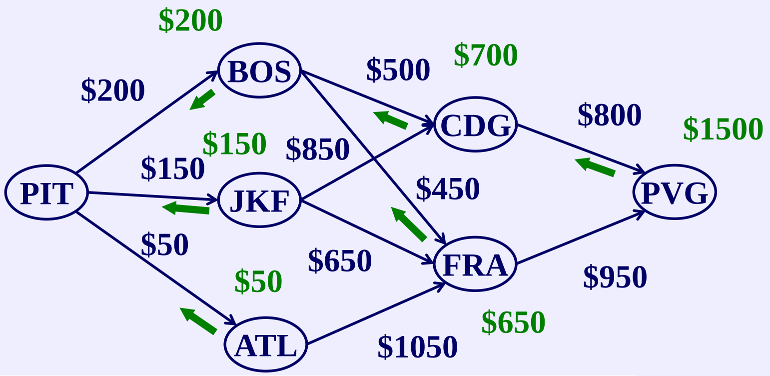
Solution: compute partial optimal, left-to-right:

Dynamic Programming Example



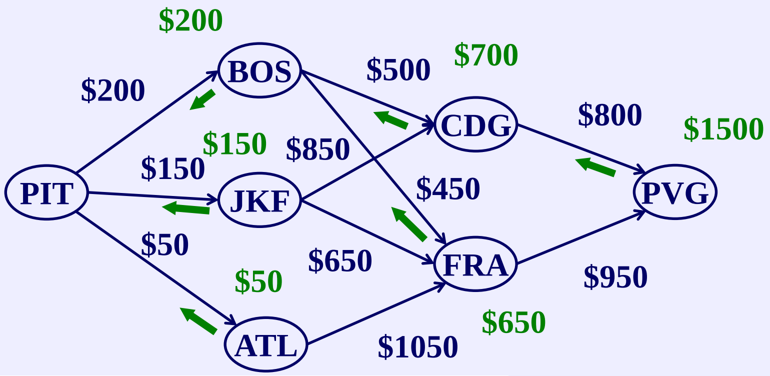
Solution: compute partial optimal, left-to-right:

Dynamic Programming Example



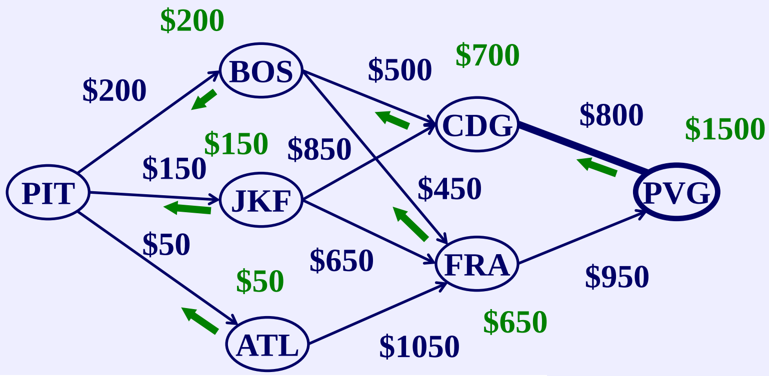
Solution: compute partial optimal, left-to-right:

Dynamic Programming Example



So, best price is $1,500 – which legs?

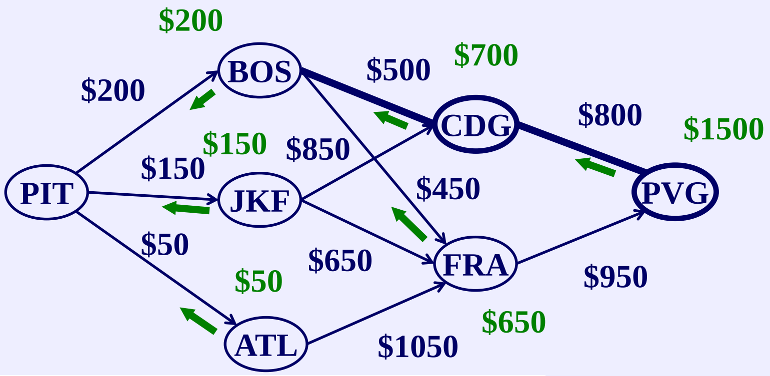
Dynamic Programming Example



So, best price is $1,500 – which legs?

A: follow the winning edges, backwards

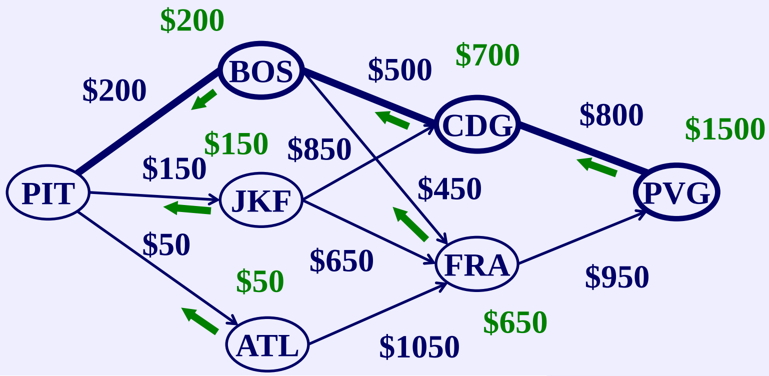
Dynamic Programming Example



So, best price is $1,500 – which legs?

A: follow the winning edges, backwards

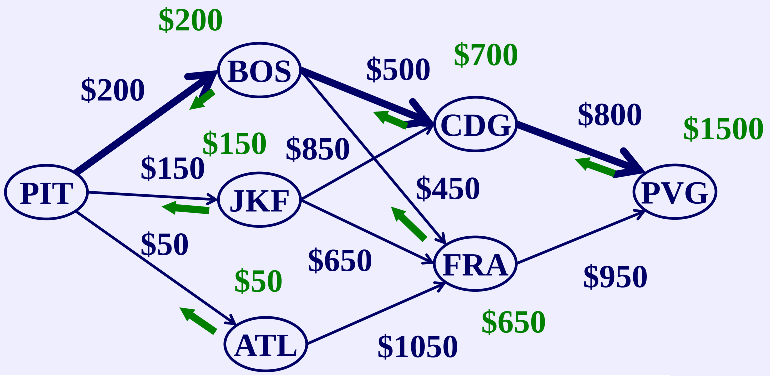
Dynamic Programming Example



So, best price is $1,500 – which legs?

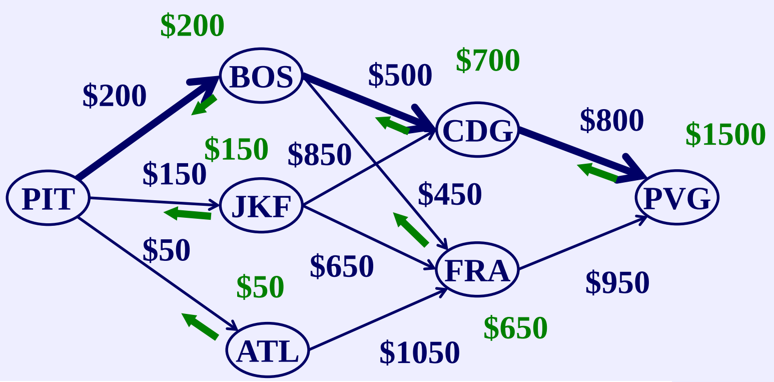
A: follow the winning edges, backwards

Dynamic Programming Example



Q: what are the states, costs and arrows, in q-opt?

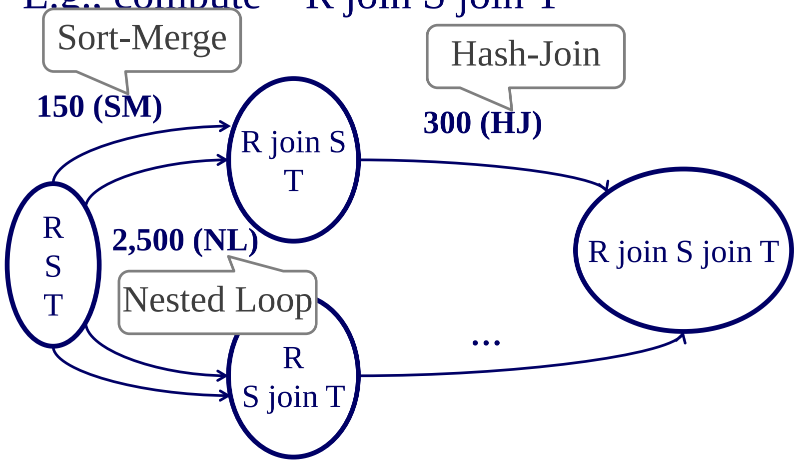
Dynamic Programming Example



Q: what are the states, costs and arrows, in q-opt?
A: set of intermediate result tables

Q-Opt + Dynamic Programming

- E.g., compute R join S join T


Sort-Merge

Hash-Join

150 (SM)

300 (HJ)

R join S T

R S T

2,500 (NL)

Nested Loop

R S join T

R join S join T

…

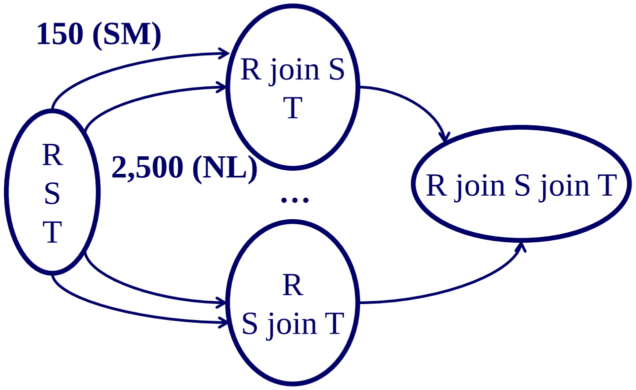
Q-Opt + Dynamic Programming

- Details: how to record the fact that, say **R** is sorted on **R.a**? or that the user requires sorted output?
- Consider the following query:

```
SELECT *
  FROM R, S, T
 WHERE R.a = S.a AND S.b = T.b
 ORDER BY R.a
```

Q-Opt + Dynamic Programming

- E.g., compute R join S join T order by R.a


150 (SM)

R join S T

R S T

2,500 (NL)

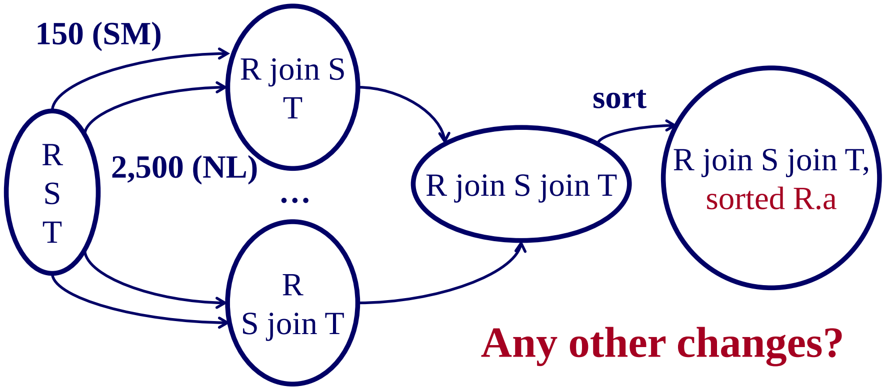
…

R join S join T

R S join T

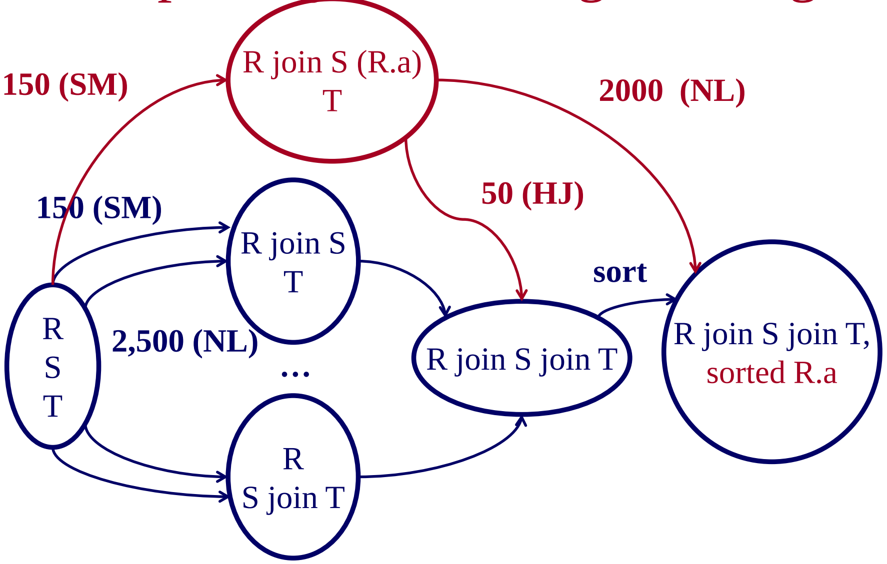
Q-Opt + Dynamic Programming

- E.g., compute R join S join T order by R.a



Any other changes?

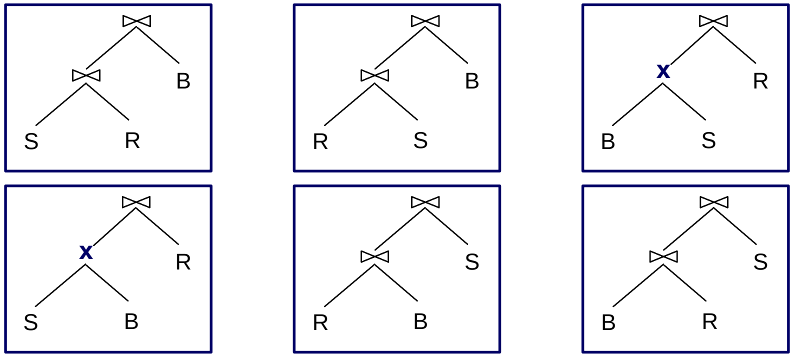
Q-Opt + Dynamic Programming



Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

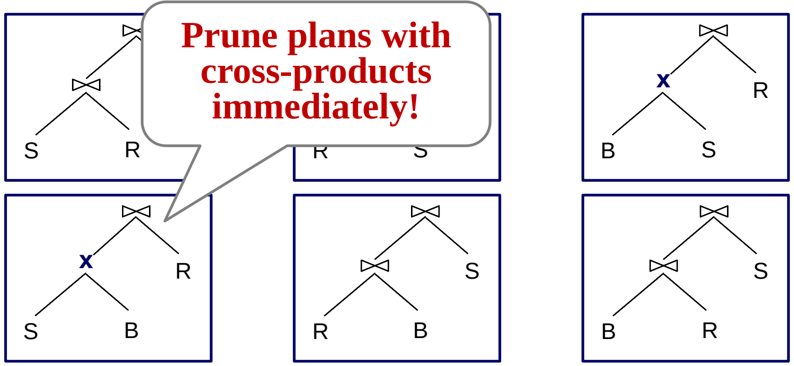
1. Enumerate relation orderings:



Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

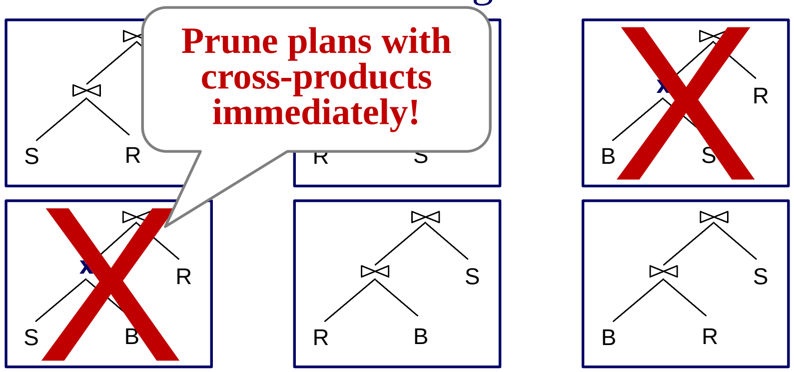


Prune plans with cross-products immediately!

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

1. Enumerate relation orderings:

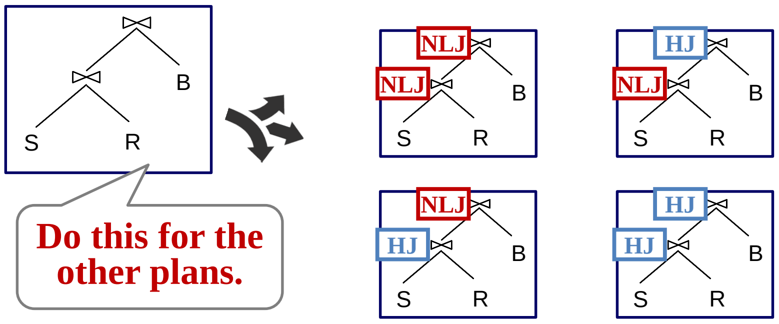


Prune plans with cross-products immediately!

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

2. Enumerate join algorithm choices:

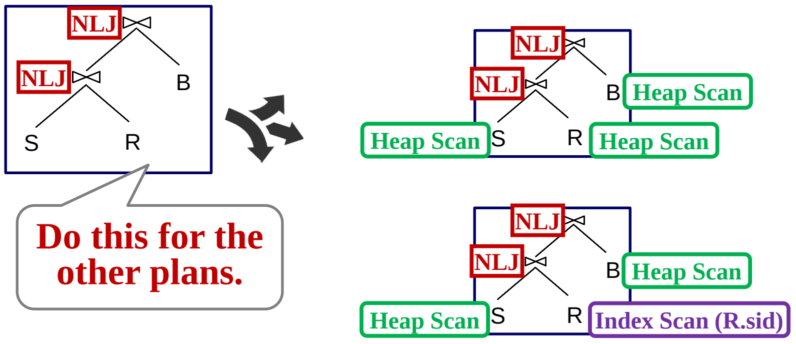


Do this for the other plans.

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

3. Enumerate access method choices:

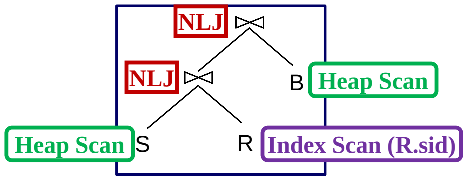


Do this for the other plans.

Candidate Plans

```
SELECT sname, bname, day
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid AND R.bid = B.bid
```

4. Now we can estimate the cost of each plan.



Query Optimization

- Bring query in internal form (eg., parse tree)
- … into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - ➡ Multiple relations.
  - Nested sub-queries.
- Estimate cost for each plan.
- Pick the best one.

Nested Sub-Queries

- Re-write nested queries
- to: de-correlate and/or flatten them

Nested Sub-Queries

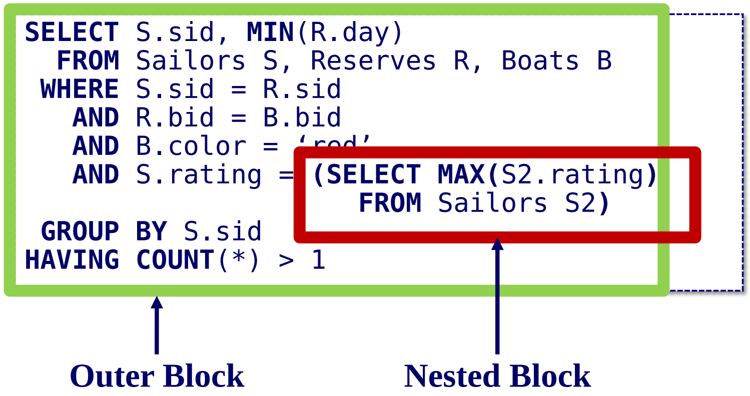
```
SELECT S.sid, MIN(R.day)
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = ‘red’
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM Sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

For each sailor with the highest rating (over all sailors) and at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.

Decomposing Queries into Blocks

- The optimizer breaks up queries into blocks and then concentrates on one block at a time.

Decomposing Queries into Blocks



```
SELECT S.sid, MIN(R.day)
  FROM Sailors S, Reserves R, Boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = ‘red’
   AND S.rating = (SELECT MAX(S2.rating)
                      FROM Sailors S2)

 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Outer Block

Nested Block

Decomposing Queries into Blocks

- The optimizer breaks up queries into blocks and then concentrates on one block at a time.
- Split *n*-way joins into 2-way joins, then individually optimize.

Query Optimizer Overview

- System R:
  - – Break query in query blocks
  - – Simple queries (ie., no joins): look at stats
  - – n-way joins: left-deep join trees; ie., only one intermediate result at a time
    - *Pros: smaller search space; pipelining*
    - *Cons: may miss optimal*
  - – 2-way joins: NL and sort-merge

Conclusions

- Ideas to remember:
  - – Syntactic q-opt – do selections early
  - – Selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - – Hash join (nested loops; sort-merge)
  - – Left-deep joins
  - – Dynamic programming