Cost-Based Optimization

Database Systems: The Complete Book

Ch 2.3, 6.1-6.4,15, 16.4-16.5

Optimizing

Optimizing

- Some equivalence rules are *always* good…
  - Which?

Optimizing

- Some equivalence rules are *always* good…
  - Which?

- Some equivalence rules are *sometimes* good
  - Which?
  - What do we do about it?

Cost Estimation

- Compare many different plans by…
  - … actually running the query
  - … estimating the plan's "cost"

Cost Estimation

THE AUTHOR OF THE WINDOWS FILE COPY DIALOG VISITS SOME FRIENDS.

Costs

Costs

- Memory Cost (Working Set Size)

Costs

- Memory Cost (Working Set Size)
- Compute Cost (“Big-O”)

Costs

- Memory Cost (Working Set Size)
- Compute Cost (“Big-O”)
- IO Cost (Pages read, Pages written)

The variable in all of these costs is the arity (size) of a relation.

How do you compute Arities?

- Heuristic Assumptions (Pick a “good enough” RF)
- Summary Statistics About The Data…
  - Upper/Lower Bounds or Value Domains
  - Distribution Summaries (Histograms)
  - Data Sampling

How do you compute Arities?

There is **no** perfect solution (yet)!

How do you compute Arities?

There is **no** perfect solution (yet)!

We don’t need a perfect solution…
… we just need one that’s good enough

Summary Statistics

- Per-Attribute Bounds / Domain Statistics
  - Assume a Uniform Distribution.
- Per-Attribute Histograms
  - Use the histogram to model the data distribution
- Data Samples
  - Use the samples to measure the RF

Uniform Distribution

$$A = 1$$

Uniform Distribution

$$A = 1$$

Chance of Hit = $^{1} / _{\text{\# of distinct values of A}}$

Uniform Distribution

$$A \in (1, 2, \ldots)$$

Uniform Distribution

$$A \in (1, 2, \ldots)$$

Chance of Hit = $|(1,2,3,\ldots)|$ / # of distinct values of A

Uniform Distribution

$$A < 3$$

Uniform Distribution

$$A < 3$$

Chance of Hit = $^{3\text{-Low}(A)} / _{\text{High}(A)\ -\ \text{Low}(A)}$

Uniform Distribution

$$\bowtie_{R.A=S.B}$$

Uniform Distribution

$$\bowtie_{R.A=S.B}$$

Chance of Hit Per B = $^{1} / _{\#\text{ Distinct Values of A}}$

Chance of Hit Per B = 1 (If B is a FK Reference)

Chance of Hit Per A = 1 (If A is a FK Reference)

Let's apply it

```
SELECT   O.Rank, COUNT(*),
FROM     Officers O
WHERE    O.Rank >= 2
  AND    O.Age > 20 AND O.Age < 30
GROUP BY O.Rank
HAVING   COUNT(DISTINCT O.Ship) > 2
```

What is the relational algebra plan for this expression?

Stats

O.Rank: 0-5 (Increments of 0.5; 11 total values)

O.Age: 16-100 (Increments of 1; 85 total values)

Officers: 40,000 tuples (over 500 pages)

Tree Indexes available over O.Age, O.Rank

What is the total cost in IOs?

What is the total cost in CPU/Tuples?

Histograms

Uniform Distributions are a strong assumption!

(data is often skewed)

Histograms

People

Name	Age	Rank
<"Alice",	21,	1 >
<"Bob",	20,	2 >
<"Carol",	21,	1 >
<"Dave",	19,	3 >
<"Eve",	20,	2 >
<"Fred",	20,	3 >
<"Gwen",	22,	1 >
<"Harry",	20,	3 >

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

VS

```
...
  AND Age = 19
```

$RF_{Age} = {}^{1}/_{nkeys} = {}^{1}/_{4}$

$RF_{Rank} = {}^{1}/_{nkeys} = {}^{1}/_{3}$

Age is best!

Histograms

People

```
 Name          Age Rank
<“Alice”,  21, 1  >
<“Bob”,    20, 2  >
<“Carol”,  21, 1  >
<“Dave”,   19, 3  >
<“Eve”,    20, 2  >
<“Fred”,   20, 3  >
<“Gwen”,   22, 1  >
<“Harry”,  20, 3  >
```

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

vs

```
…
  AND Age = 19
```

$RF_{Age\text{-}20} = {}^{1}/_{2}$

$RF_{Rank} = {}^{1}/_{3}$

Age is worst!

Histograms

People

```
 Name         Age Rank
<“Alice”,   21, 1  >
<“Bob”,     20, 2  >
<“Carol”,   21, 1  >
<“Dave”,    19, 3  >
<“Eve”,     20, 2  >
<“Fred”,    20, 3  >
<“Gwen”,    22, 1  >
<“Harry”,   20, 3  >
```

```
SELECT Name
FROM People
WHERE Rank = 3
  AND Age = 20
```

vs

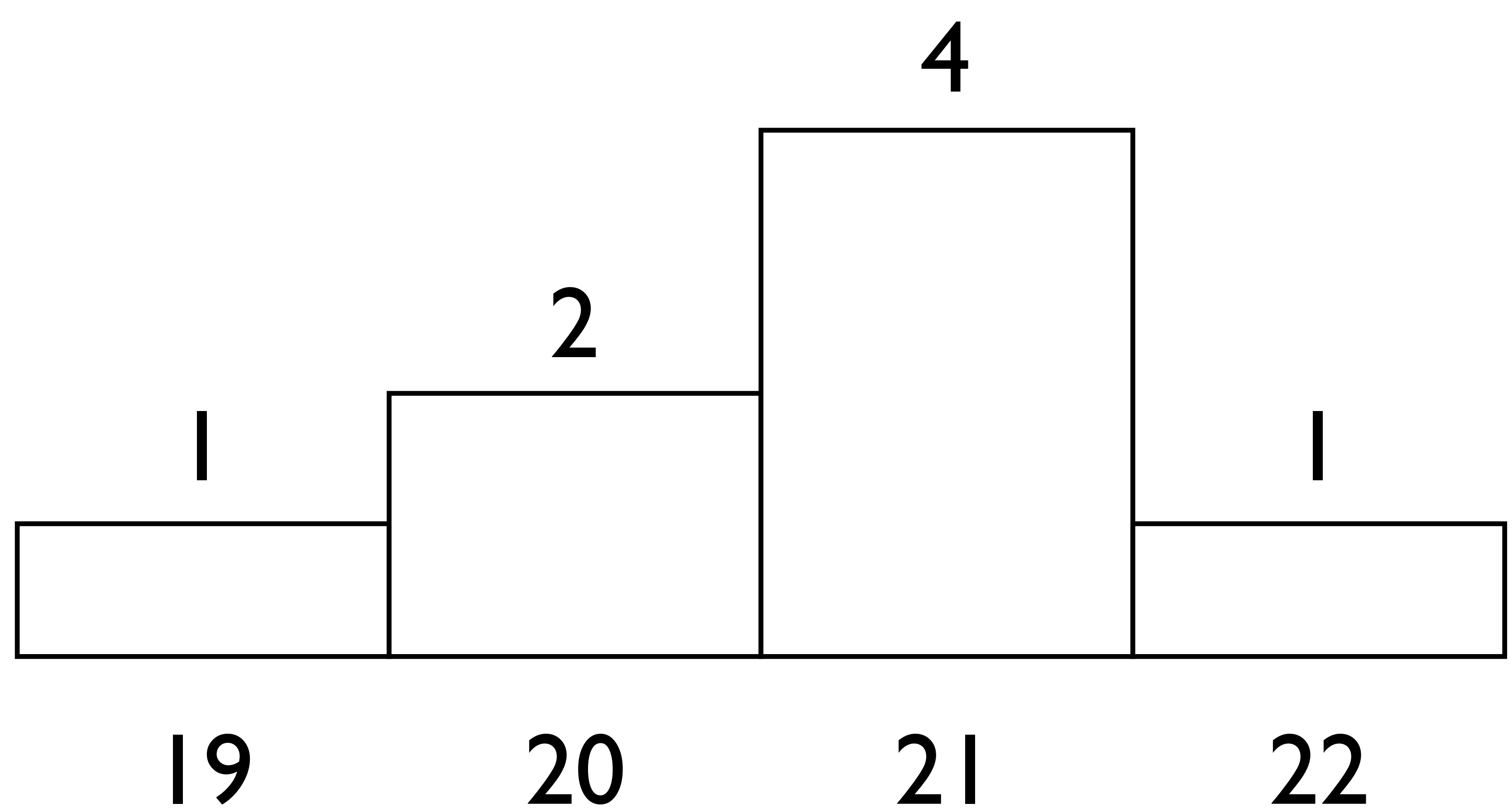
```
…
  AND Age = 19
```

$RF_{Age\text{-}19} = {}^{1}/_{8}$

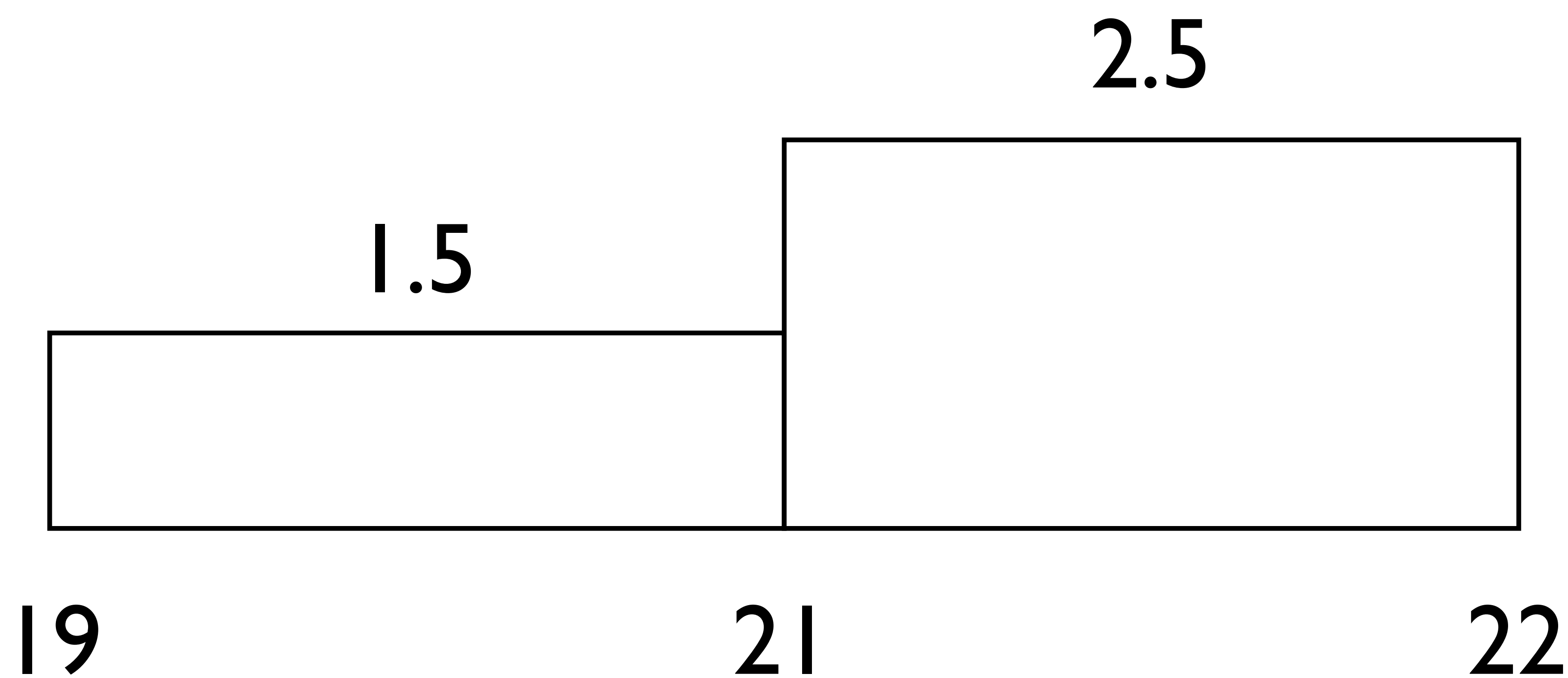
$RF_{Rank} = {}^{1}/_{3}$

Age is best!

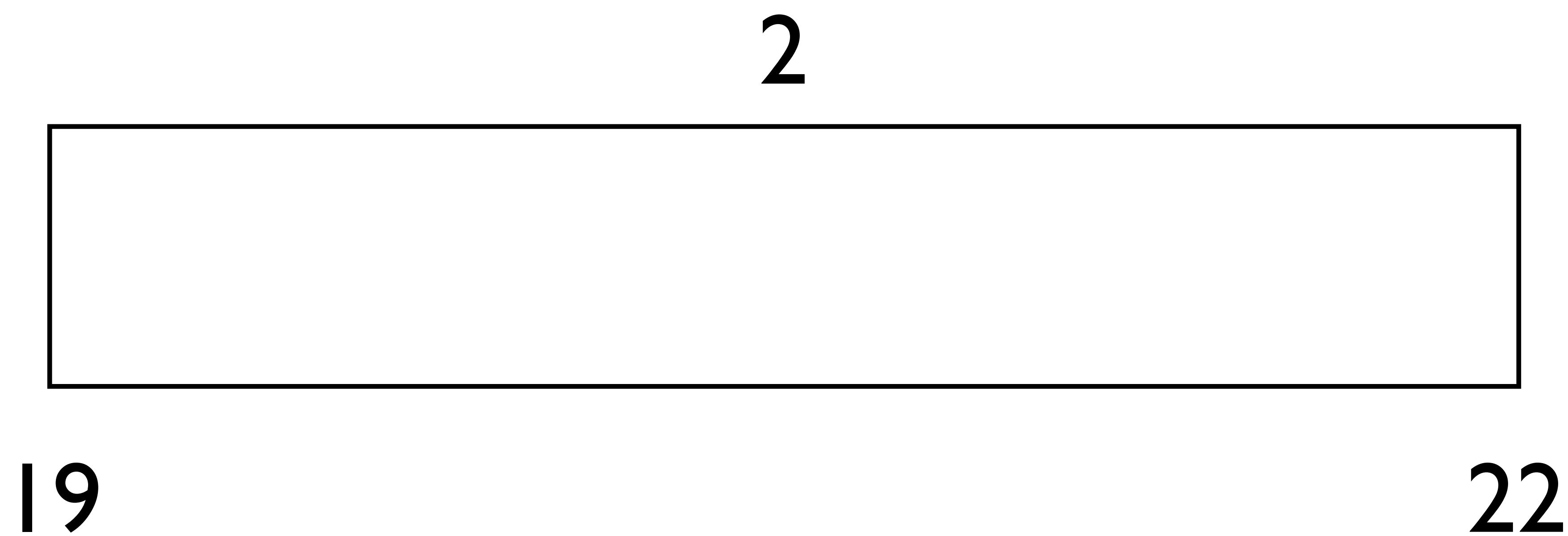
Histograms



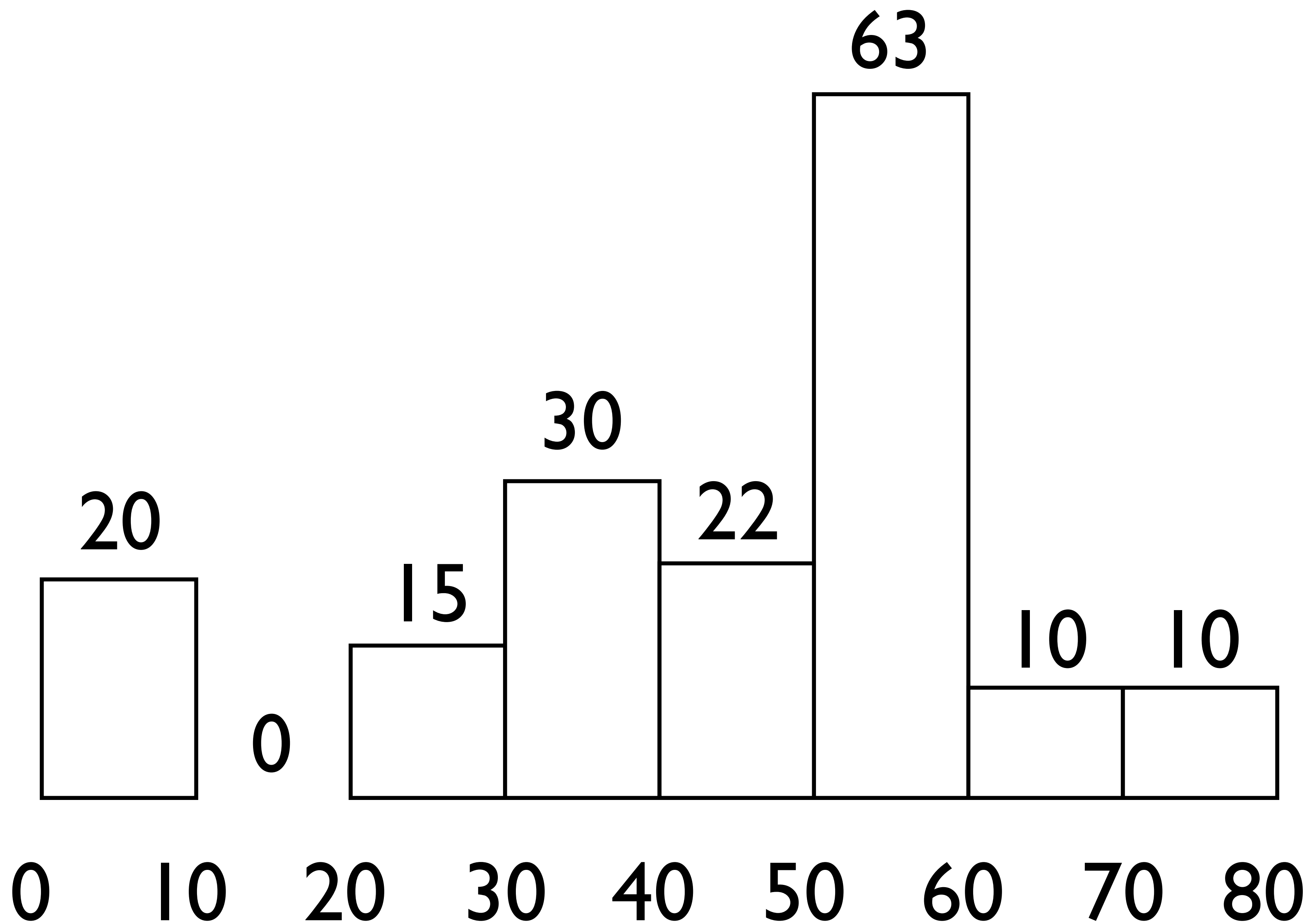
Histograms



Histograms

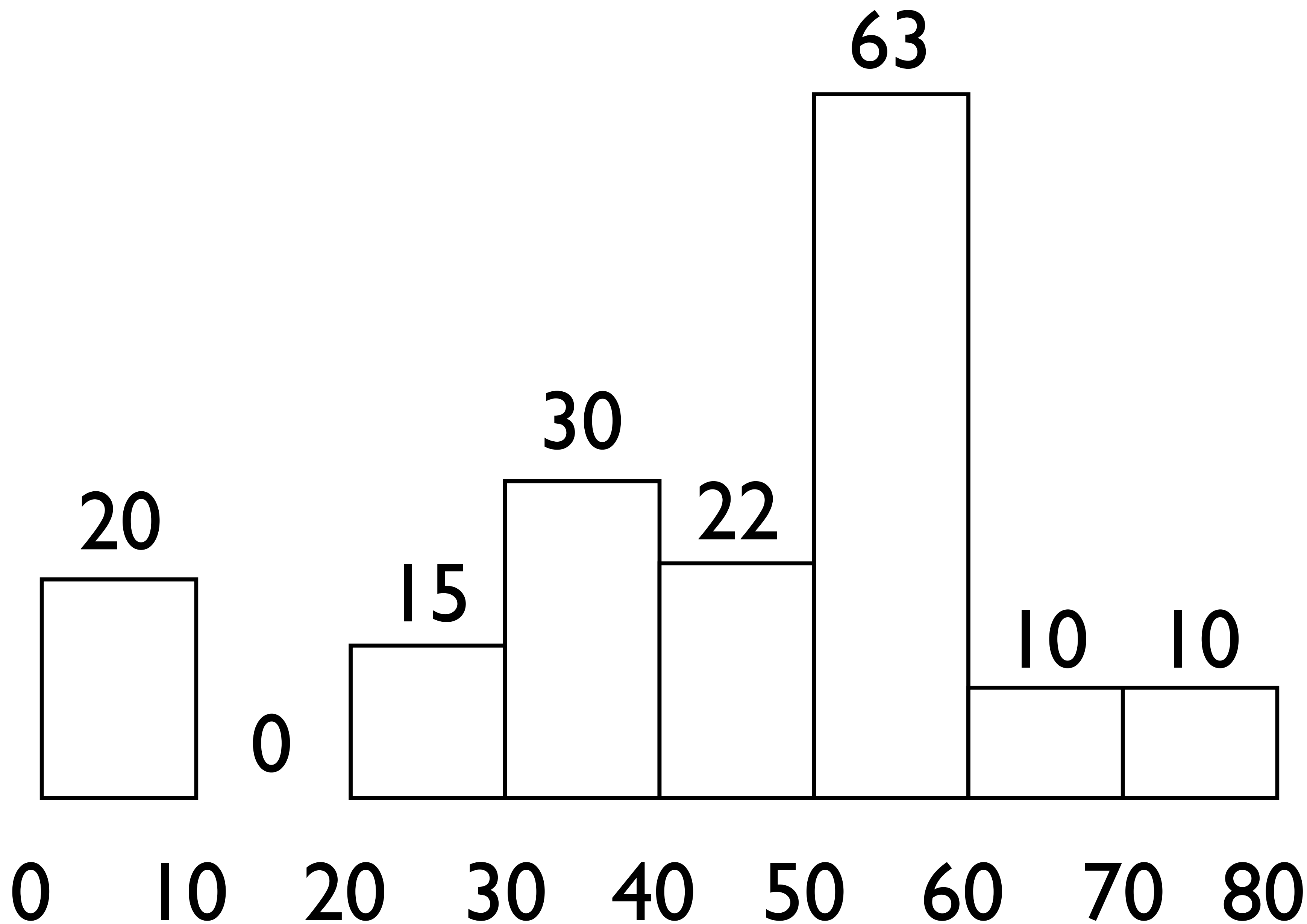


Histograms



`SELECT … WHERE A = 33`

Histograms




```
SELECT … WHERE A > 33
```

Using Constraints

- A Key attribute has one distinct value per row (equality selects exactly one row)
- Foreign Key joins generate one row for each row in the **referencing** relation.
- Cascade relationship guarantees EXACTLY one row per reference.

Sampling

- Take a bunch of tuples from each relation.
- Run 2-3 different query plans on these tuples.
  - Estimate the sampling factors for each operator in the plan based on how many survive.

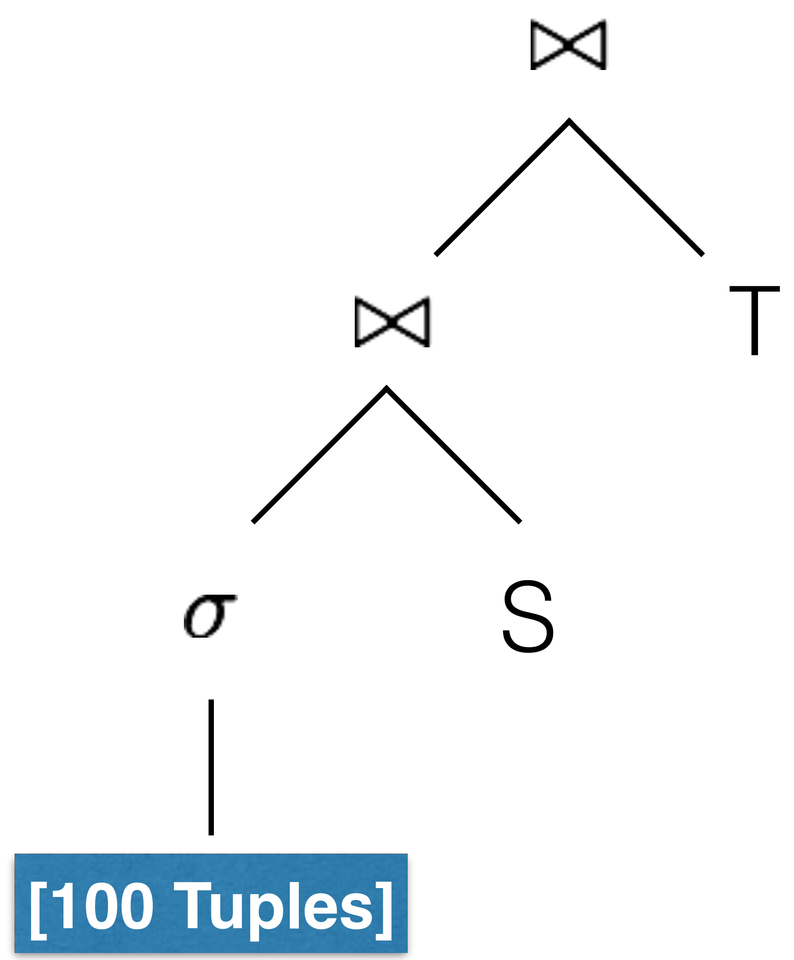
Sampling

How big is a “bunch?”

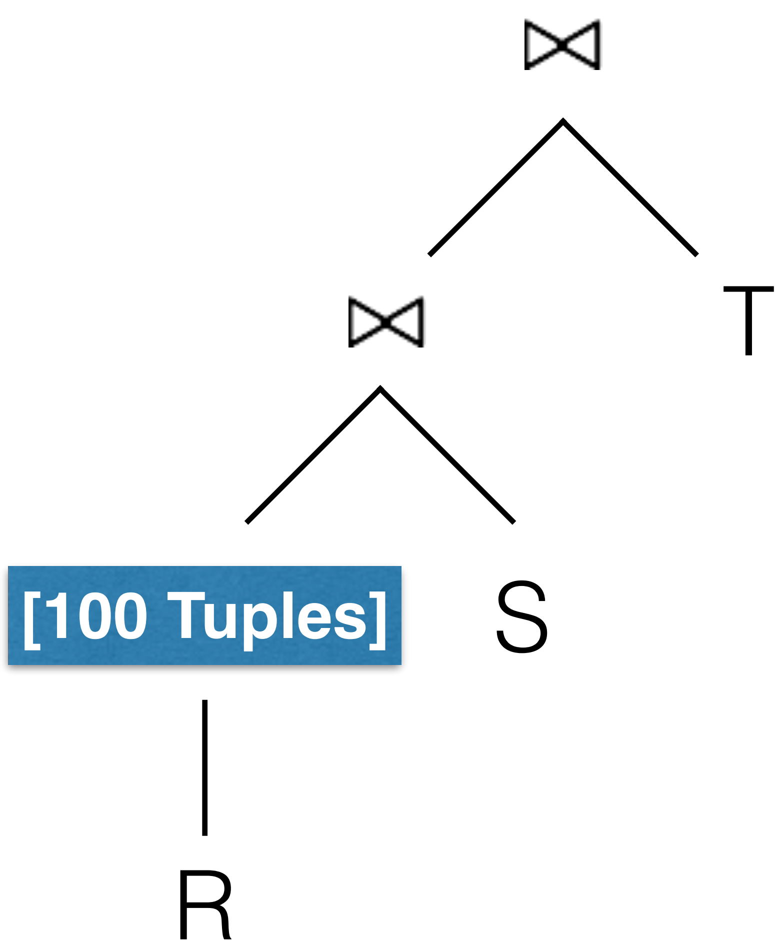
Sampling

- **Problem**: Very Selective Predicates
- **Problem**: Joins and the Birthday Paradox
- **Problem**: Counting Aggregate Groups

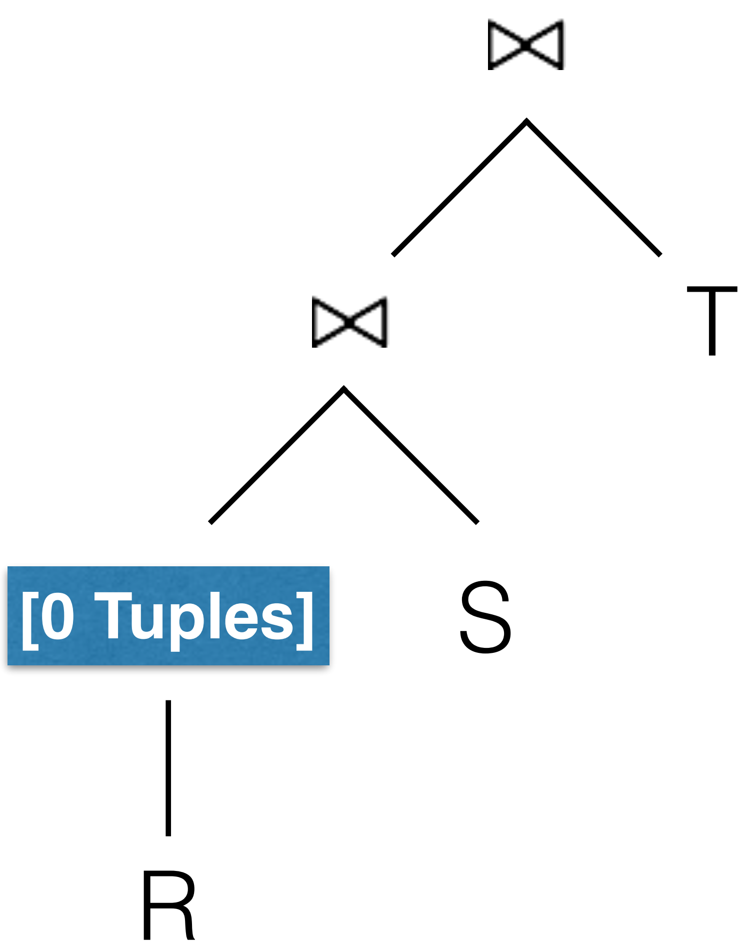
Very Selective Predicates



Very Selective Predicates



Very Selective Predicates



Join Conditions

Image: Wikipedia

Join Conditions

<u>Birthday Paradox</u>

Need $O(\sqrt{|R|+|S|})$ tuples to reliably guess RF for equijoin

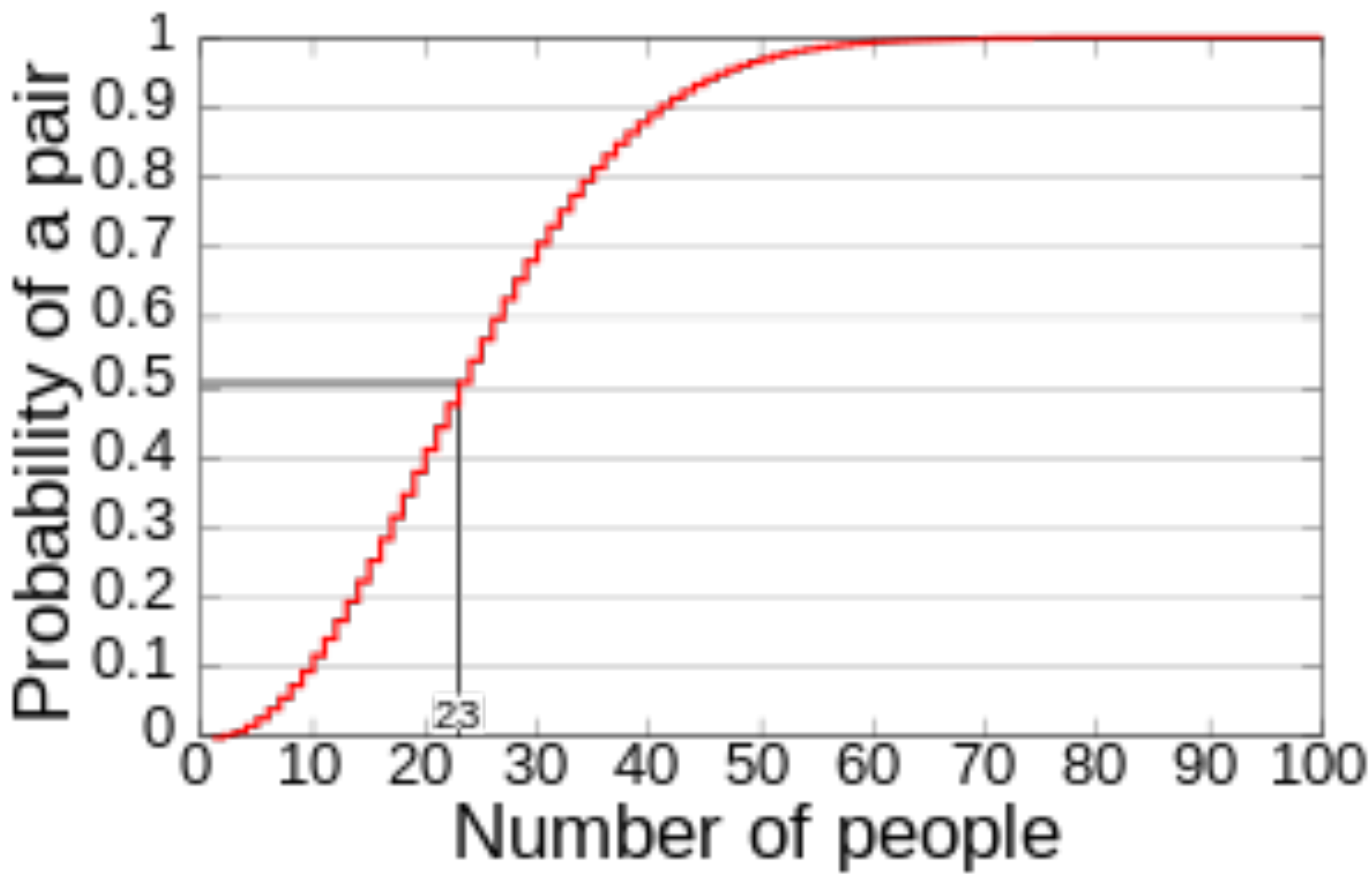


Image: Wikipedia

Estimating Join Costs

How many query plans are there?

$$R \bowtie S \bowtie T \bowtie U$$

Estimating Join Costs

There are (N-1)! (factorial) different ways (plans) to evaluate this join.

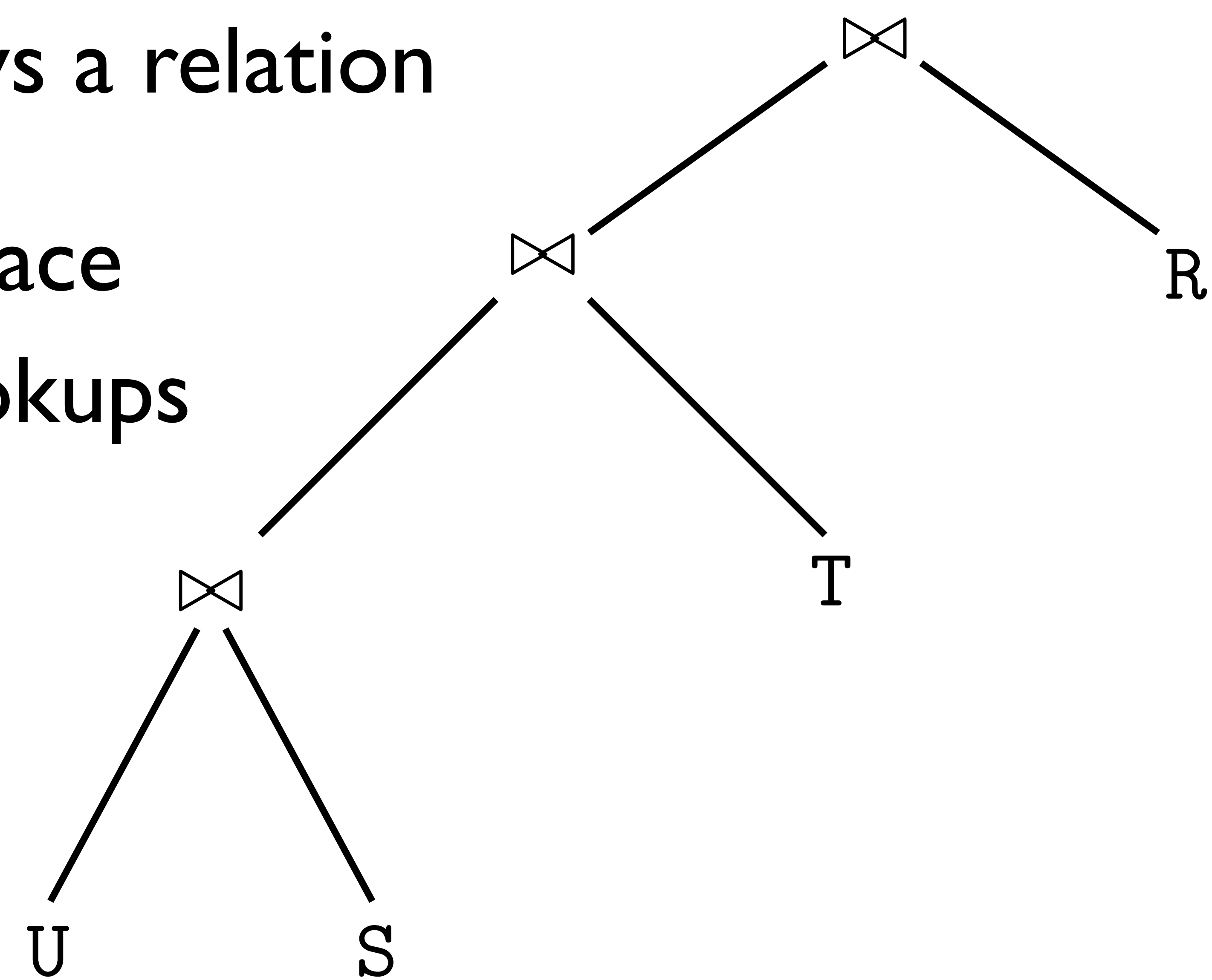
Computing costs for all of these plans is expensive!

Left-Deep Plans

RHS Join Input is always a relation

1) Shrinks join search space
2) Allows index scans/lookups

Technique Pioneered by
the System R Optimizer



In Practice

Heuristics, Histograms and Sampling are "good enough" to optimize the common cases.

In Practice

Heuristics, Histograms and Sampling are "good enough" to optimize the common cases.

Some relational databases have manual overrides.

Oracle

```
SELECT /*+ INDEX (employees emp_department_ix)*/
       employee_id, department_id
  FROM employees
  WHERE department_id > 50;
```

Postgres

```
SELECT attname, inherited, n_distinct,
       array_to_string(most_common_vals, E'\n') as most_common_vals
FROM pg_stats
WHERE tablename = 'road';
```

```
 attname | inherited | n_distinct |          most_common_vals
---------+-----------+------------+------------------------------------
 name    | f         |  -0.363388 | I- 580                        Ramp+
         |           |            | I- 880                        Ramp+
         |           |            | Sp Railroad                       +
         |           |            | I- 580                            +
         |           |            | I- 680                        Ramp
 name    | t         |  -0.284859 | I- 880                        Ramp+
         |           |            | I- 580                        Ramp+
         |           |            | I- 680                        Ramp+
         |           |            | I- 580                            +
         |           |            | State Hwy 13                  Ramp
```

In Practice

Heuristics, Histograms and Sampling are "good enough" to optimize the common cases.

In Practice

Heuristics, Histograms and Sampling are "good enough" to optimize the common cases.

Some relational databases have manual overrides.

In Practice

Heuristics, Histograms and Sampling are "good enough" to optimize the common cases.

Some relational databases have manual overrides.

All relational databases have an "EXPLAIN" operator

Postgres

```
EXPLAIN SELECT sum(i) FROM foo WHERE i < 10;

                          QUERY PLAN
----------------------------------------------------------------------
 Aggregate  (cost=23.93..23.93 rows=1 width=4)
   ->  Index Scan using fi on foo  (cost=0.00..23.92 rows=6 width=4)
         Index Cond: (i < 10)
```

Backup Slides

Join Algorithm Comparison

	Can Support Pipelining?	But?
Hybrid Hash	**Yes**	RHS Hash Table needs to fit in memory
Index Nested Loop	**Yes**	RHS Table needs an index on the join key
Sort/Merge Join	**Yes**	LHS and RHS must both be sorted on the join key
(Block) Nested Loop	**Yes**	RHS Table needs to fit in memory
Hash Join	No	No buts. Hash Join always materializes

Join Algorithm IO Costs

$R \bowtie S$	IO Cost
Hybrid Hash	[#pages of S] (if fits in mem)
Index Nested Loop	|R| * [cost of one scan/lookup on S]
Sort/Merge Join	[#pages of S] (+sorting costs)
Nested Loop	[#pages of S] (if fits in mem)
Block Nested Loop	[#pages of R] + [#of block pairs] * ([#pages per block of R]+[#pages per block of S])
Hash Join	2*([#pages of R]+[#pages of S]) + [#pages of S]

Data Access IO Costs

	Full Scan	Range Scan	Lookup
Raw File	N	N	N
Sorted File	N	$\log_2(N)+\|R\|$	$\log_2(N)$
Static Hash Index	>N	>N	~1
Extendible Hash Index	>N+|D| (random)	>N+|D| (random)	2
Linear Hash Index	>N	>N	~1
ISAM Tree Index	~N	$\sim\log_{\|T\|}(N)+\|R\|$	$\sim\log_{\|T\|}(N)$
B+ Tree Index	N (random)	$\log_{\|T\|}(N)+\|R\|$ (random)	$\log_{\|T\|}(N)$