PRECOMPUTATION

Optimization Recap

- There are alternative ways of evaluating a query
  - Equivalent expressions
  - Different algorithms for each operation
- Some equivalence rules are always good (Which?)

Optimization Recap

- There are alternative ways of evaluating a query
  - Equivalent expressions
  - Different algorithms for each operation
- Some equivalence rules are always good (Which?)
  - Selection pushdown, Join conversion, Projection pushdown, Eliminating redundant distinct operators, Eliminating redundant sort operators

File Scan

- What is the cost?
  - Number of I/Os: |R|
- What is the output size?
  - Size of the relation: |R|

Index Scan (with Selection Condition)

- What is the cost?
  - Number of I/Os : $|\sigma(R)|$
- What is the output size?
  - $|\sigma(R)|$

Index Scan (with Selection Condition)

- If you index the value locations of a column that ranges between 1-10
  - 1 -> 2,3,6,15,22,23,27
  - 2 -> 1,4,7,8,9,…
  - 3 -> 5,10,11,14,…
  - …
- You can join that column with a PK very efficiently
- You can filter values more efficiently
- Guess/Compute the size of the output right away

Index Scan (with Selection Condition)

- If you index the locations of a PK column that ranges between 1-10
  - 1 -> 2,3,6,15,22,23,27
  - 2 -> 1,4,7,8,9,…
  - 3 -> 5,10,11,14,…
  - …
- You can join with a PK very efficiently

Index Scan (with Selection Condition)

- Namely
  - Compute Min(R.a)
  - Compute Max(R.a)
  - Index the locations of each value in (R.a)
  - You can create buckets/partition data
  - You can create hashtables of selected data

Projection

- What is the cost?
  - Number of I/Os: 0
- What is the output size?
  - |R|

Selection

- What is the cost?
  - Number of I/Os: 0
- What is the output size?
  - $|\sigma(R)|$

Cardinality (Size) Estimation

- Most of the operators are straightforward
  - $\pi(R)$, $\tau(R)$: $|R|$
  - $R \cup S$: $|R| + |S|$
  - $R \times S$: $|R| * |S|$
  - $R \bowtie S$: Identical to $\sigma(R \times S)$
- Some are hard, why?
  - $\sigma(R)$
  - $\gamma(R)$, $\delta(R)$

Computing Selectivity ($|\sigma(R)| = ?$)

- Generic (Default) Heuristic
  - Selectivity = Half of the tuples
  - Is it correct?
  - Does it work?
  - Some databases actually use this heuristic

Computing Selectivity ($|\sigma(R)| = ?$)

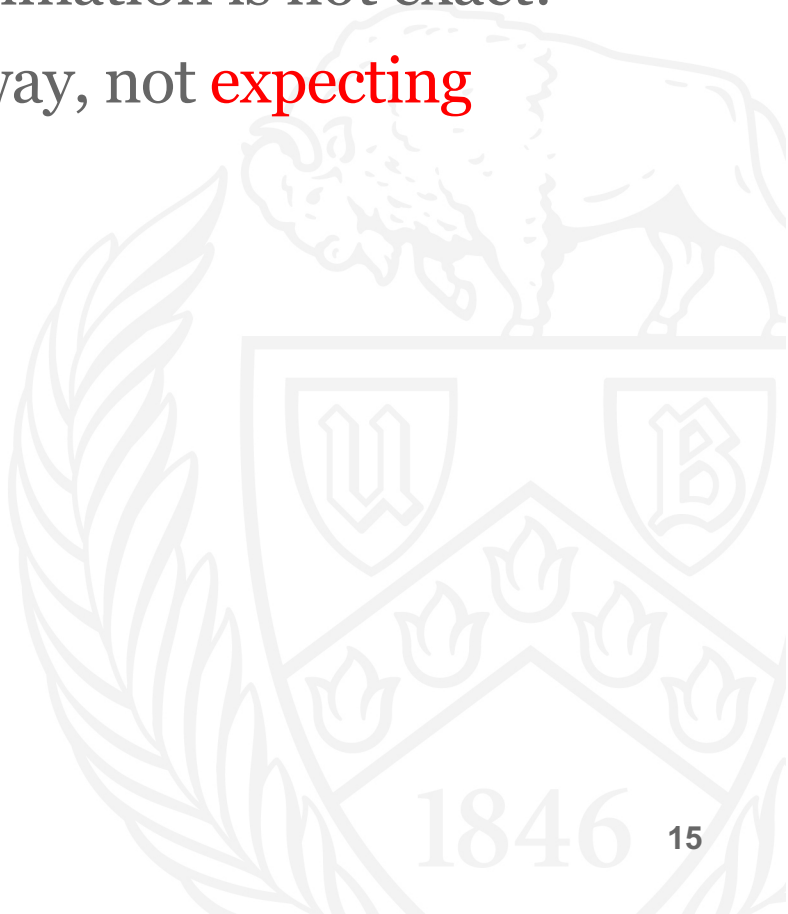
- R.a = [Constant]
  - If R.a is a key, then precisely 1 tuple satisfies the condition
- Idea: Collect stats on # of distinct values
  - Selectivity = 1 / # of distinct values of R.a
  - Works well on discrete data
- Can you do even better?
  - Yes you can! Histograms!

Computing Selectivity of a Join

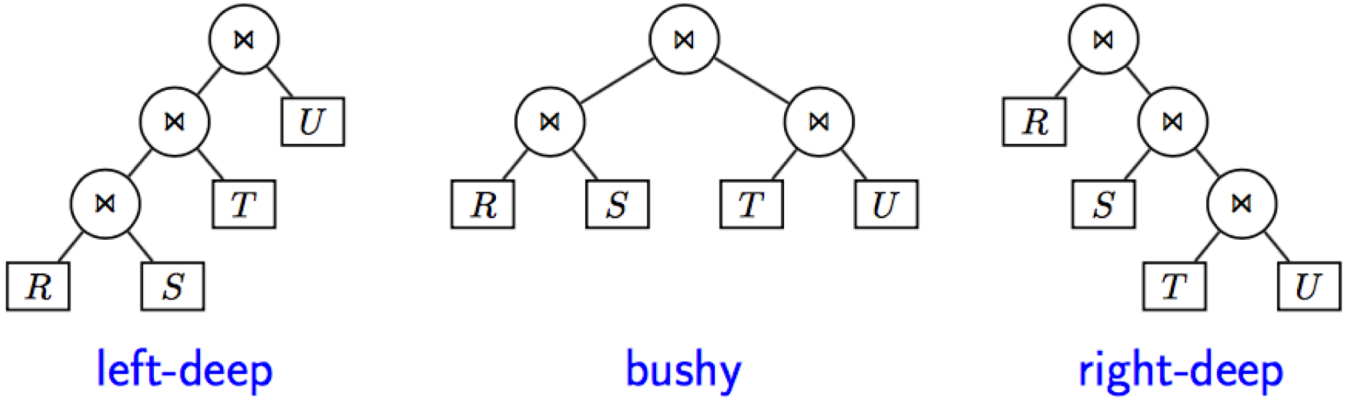
- R.a = S.b
- Idea: Assume no correlation
  - Becomes identical to either R.a = [const] or S.b = [const]
  - For each row, you are testing whether S.b = some specific, somewhat arbitrary value
  - Both are an upper bound on the selectivity, so take whichever reduction gives you the lower value

Cardinality (Size) Estimation

- We should be clear by now: Size estimation is not exact!
- We are trying to estimate the best way, not expecting exact results out of it
- Smaller cardinality is good!



Join Ordering



In practice a query compiler usually only considers left-deep join orderings.

- There are still n! possible orderings of this form, but that is already a lot less.
- Left-deep orderings use, in general, less memory. Furthermore, in general, they require fewer subresults to be stored.

Plan Selection

To compute the best physical plan for a given logical query plan, we should, in principle:

1) Calculate all possible (left-deep) join orderings of the logical plan
2) For each such plan calculate all possible assignments of physical operators to the nodes
3) From this enormous pile of candidate physical query plans, choose the one with the least estimated cost

Query compilation MUST NOT take longer than the actual execution of the query

In a complex query, it can be impossible to inspect all candidate physical plans.

Heuristics! Branch-and-Bound Plan Enumeration, Hill climbing, Dynamic programming, Slinger-Style, Greedy

Greedy Ordering

- Greedy heuristics produce suitable join trees very fast
- Suitable for large queries
- It produces a sequence of relations, so it uniquely identifies the left-deep join tree
- Idea: minimize output volumes

Greedy Ordering

- Start with a logical query plan without join ordering
- We work bottom-up: first we assign physical operators to the leaves, then to the parents of the leaves, then to their parents, and so on. At each point we choose the physical operator with the least cost.
- When we reach a join operator, we need to determine an ordering of its various members:

  Next slide

Greedy Ordering

- We start by joining the two relations for which the best physical join algorithm yields smallest cost
- Add, from the remaining relations, add another join on top of them that yields the smallest cost
- Repeat the previous step until we add all the join orderings

Example

- $R \bowtie T = 100$
- $R \bowtie S = 200$
- $R \bowtie U = 300$
- $T \bowtie S = 150$
- $T \bowtie U = 400$
- $U \bowtie S = 500$

- What is the ideal $R \bowtie T \bowtie S \bowtie U$ order?

Example

- $R \bowtie T = 100$
- $R \bowtie S = 200$
- $R \bowtie U = 300$
- $T \bowtie S = 150$
- $T \bowtie U = 400$
- $U \bowtie S = 500$

- $(((R \bowtie T) \bowtie S) \bowtie U)$

Greedy Ordering

- It doesn't need to return the optimal plan, but it will be close to ideal.

Histograms

- Uniform distributions are a strong assumption!

```
SELECT name
FROM People
WHERE rank = 3
AND age = 20
```

vs

```
AND age = 19
```

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

Histograms

Reduction Factor

(Assuming uniform distribution)

$RF_{AGE} = 1/4$

$RF_{RANK} = 1/3$

Age is better…

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

Histograms

Reduction Factor

(Actual distribution)

$RF_{AGE-20} = 1/2$

$RF_{AGE-19} = 1/8$

$RF_{RANK-1} = 3/8$

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

Sampling

- You can't always collect comprehensive statistics on the data
- 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 rows are outputted.