CS186 Discussion Section Week 8

Query Optimization

Today

1. Review of Optimization Goals
2. Histograms
3. System R (Selinger-style) - optimizer review
4. More time for practice problems

Review of Optimization Goals

- Why optimize?
  - Better resource utilization.
  - Faster queries. (For some queries, easy to get 99% reduction in query time!)
- What do we optimize?
  - Traditionally - total I/Os + f*CPUs to execute query.
  - Could optimize for time to first answer, power consumption, etc.
- What enables optimization?
  - Catalog
  - Relational Algebra
  - Operator model

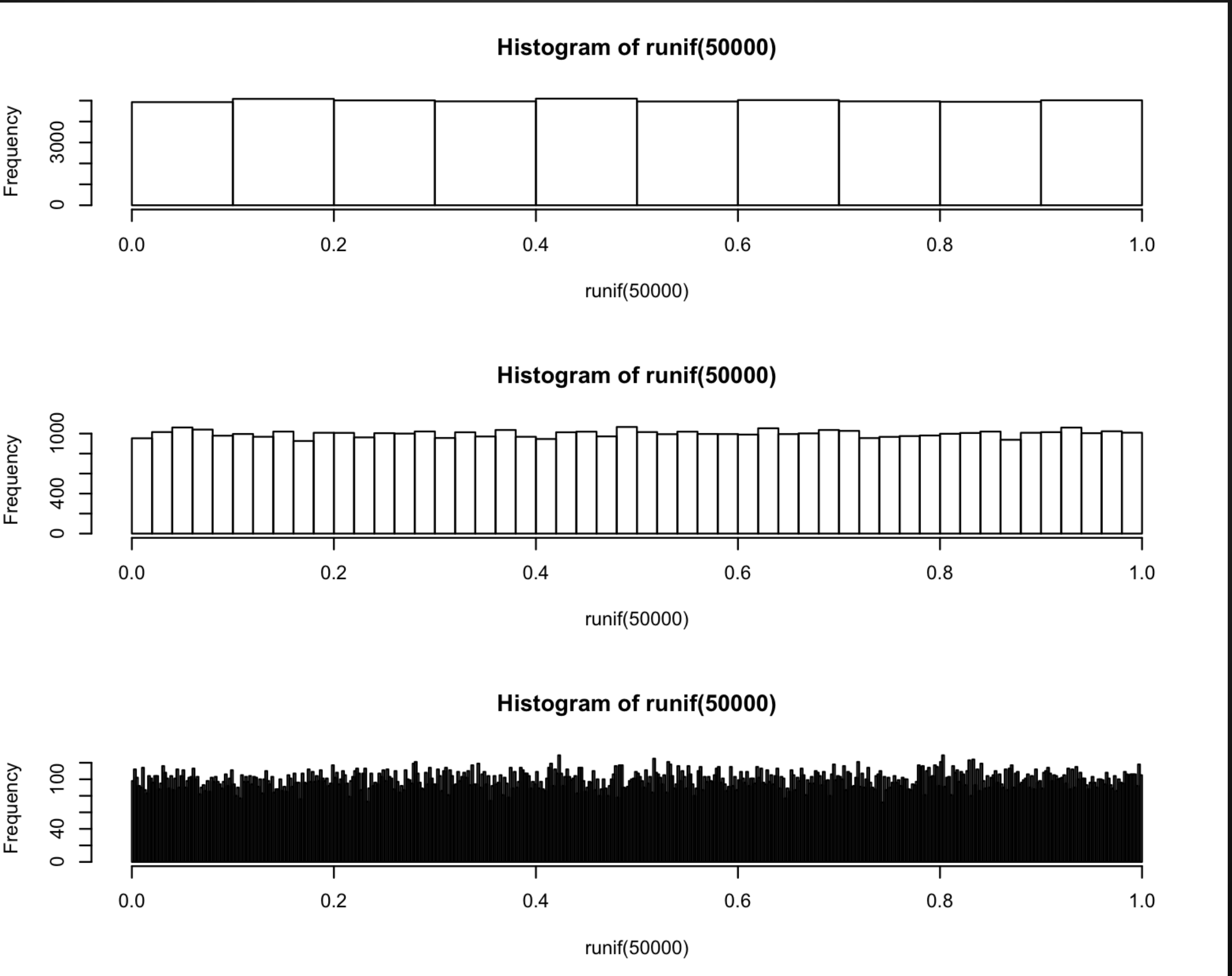
Histograms

"... an estimate of the probability distribution of a continuous variable" - Wikipedia

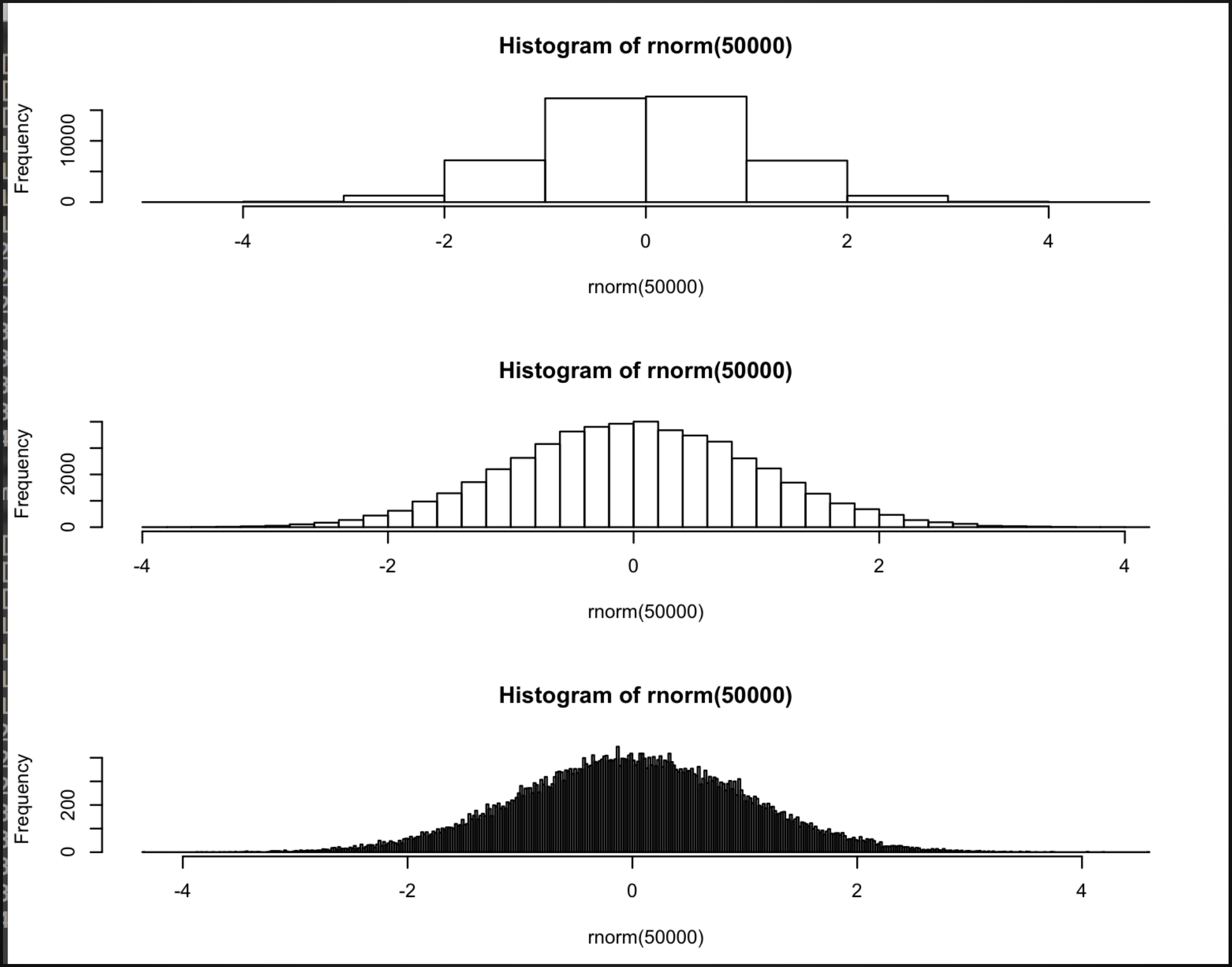
Often a visual tool - in databases we use them as a concise representation of the distribution of one or more attributes.

Gives us a better estimate of selectivity than "assume things are uniform."

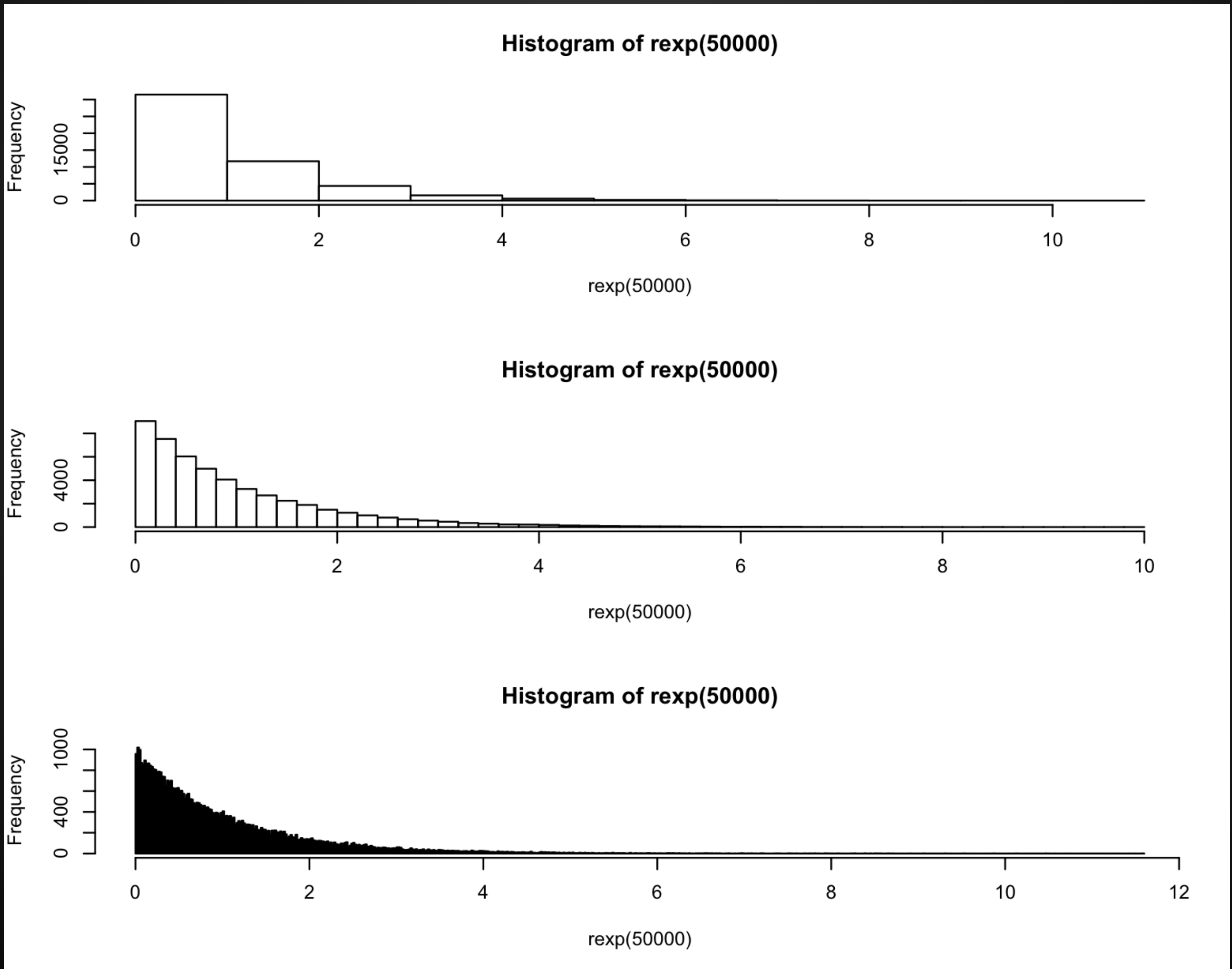
Histogram Examples - Uniform



Histogram Examples - Normal



Histogram Examples - Exponential



Histogram Wrapup

- Concise representation of data distribution.
- Useful for selectivity estimation.
- Think of it as lossy compression.
- More buckets:
  - better estimates
  - more storage
- You’ll implement these!

Query Optimization - Selinger Style

- System R Optimizer
- Cost model
  - Minimize: I/Os + f*CPUs
- Estimate cost:
  - Estimate cost for each operator - sum them up!
  - Requires selectivity estimates
    - 1/10 if not available!
  - “Interesting Orders” change plan cost
- Prune the search space
  - Left-deep plans only!
- Search the space
  - Dynamic programming!

Cost Model

A query plan has a single number associated with it - its cost:

COST = IOs + f*CPUs

One number allows us to say “this plan is better than that plan.”

f - A factor that we can set to determine which is more important - IOs or CPUs.

Determinants of Plan Cost

- Access method of base tables
  - Scan
  - Index
    - Range vs. lookup
    - Clustered vs. unclustered
- Join ordering
  - Do we want to keep rereading a big table over and over again?
  - What if we have a highly selective scan?
- Join method
  - Sort-merge? Hash? BNL?

Left Deep Plans

An optimization that accomplishes a few things

1. Prunes search space of possible join orderings from something like *n!Catalan[n-1]* ≈ $n!4^n$ to something more like *n!*
    a. See Catalan numbers.
    b. #(Left deep plans) << #(All Plans)
2. Gives us all “fully pipelined” plans
    a. Also gives us some plans that are not fully pipelined.

Note: number of possible plans is still *n!* !!!

Dynamic Programming

1. Find the best 1-table access method.
2. Given the best 1-table method as the outer, find the best 2-table.
3. …
4. Given the best (N-1)-table method as the outer, find the best N-table.

*Wrinkle - instead of “strictly the best” we return the best for each interesting order of the tuples.

*Wrinkle 2 - do cross products last!

Interesting Orders

Operator returns an “interesting order” if its result is in order of:

- some *ORDER BY* attribute
  - means we don’t have to sort later!
- some *GROUP BY* attribute
  - means we can use the nice scan method for our group-by later!
- some Join attribute of *other* joins
  - Means we can use sort-merge far cheaper!

Now - practice!