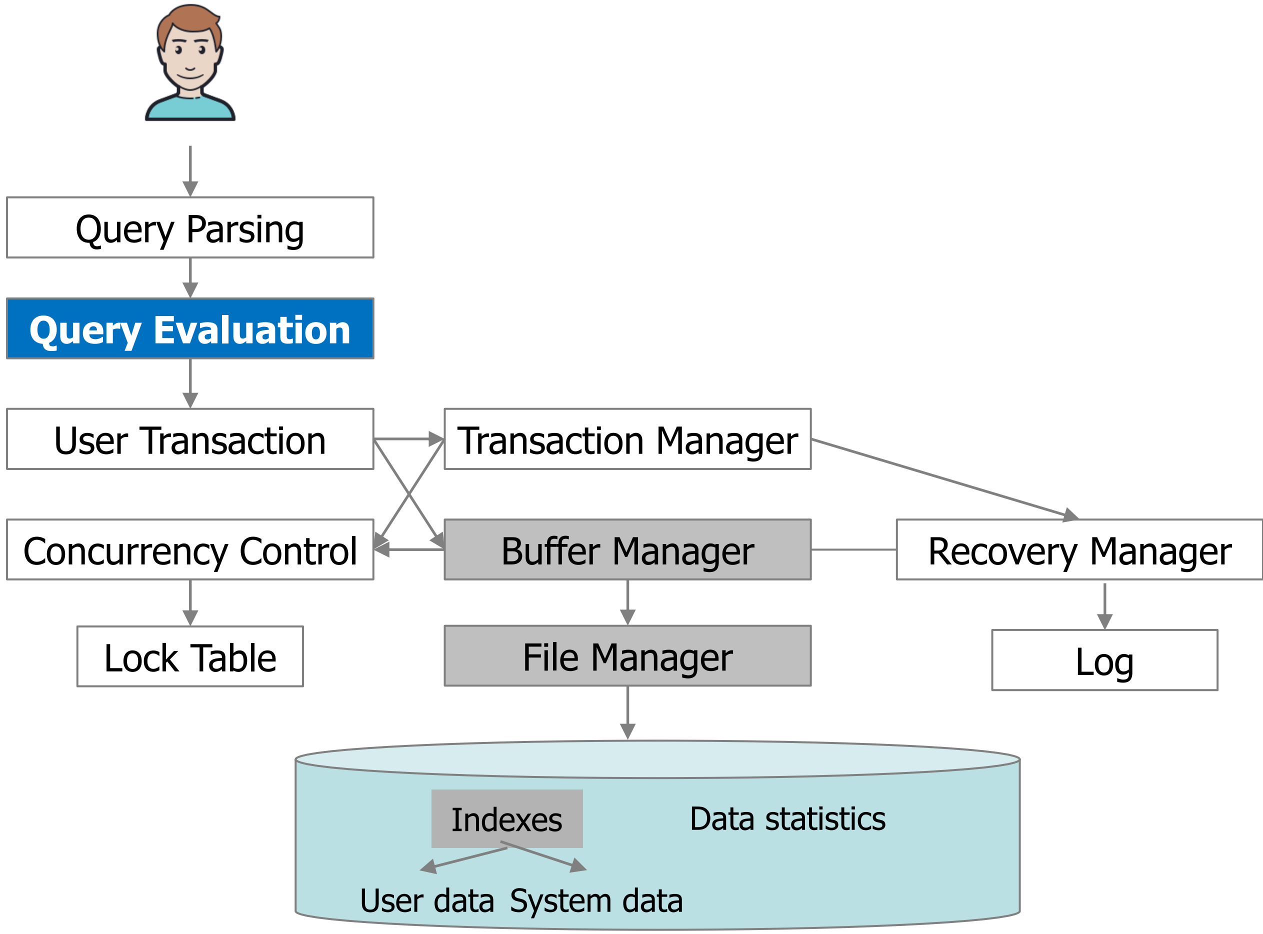
M146 Database Systems

Spring 2020

Georgia Koutrika

Query Evaluation:



Next Topics

1. Index Selection
2. Histograms
3. Logical Optimization
4. Physical Optimization

Indexes

While we all know that the Query Optimizer uses Indexes to produce better execution plans, we don't all know exactly which indexes will give the best results.

There are two questions:

Problem 1: Which indexes we should create at the first place

Problem 2: Which indexes (from the ones created) the Query Optimizer will choose

If we understand how the Query Optimizer chooses indexes, then we will know which indexes to create in the first place

Problem 1: Which indexes to create

Input:

- Schema of the database
- **Workload description:** set of (query template, frequency) pairs

Goal: Create a set of indexes that minimize execution time of the workload.

- Cost / benefit balance: Each additional index may help with some queries, but requires updating

This is an optimization problem!

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category = ?
```

Frequency
10,000,000

```
SELECT pname,
FROM Product
WHERE year = ? AND Category = ?
AND manufacturer = ?
```

Frequency
10,000,000

Which indexes might we choose?

Example

Workload description:

```
SELECT pname
FROM Product
WHERE year = ? AND category =?
```

Frequency
10,000,000

```
SELECT pname
FROM Product
WHERE year = ? AND Category =?
AND manufacturer = ?
```

Frequency
100

Now which indexes might we choose?
Worth keeping an index with manufacturer in its search key around?

How to select the indexes?

- Can be framed as standard optimization problem: Estimate how query cost changes when we add index.
    - We can ask the optimizer!

- Search over all possible space is too expensive, optimization surface is really nasty.
    - Real DBs may have 1000s of tables!

- Techniques to exploit *structure* of the space.

NP-hard problem, but can be solved!

Currently, all major commercial database vendors include a physical database design tool to help with the creation of indexes.

E.g., Oracle Autonomous DB

Automatic indexing

The automatic indexing feature automates the index management tasks in an Oracle database.

Automatic indexing automatically creates, rebuilds, and drops indexes in a database based on the changes in application workload, thus improving database performance.

The automatically managed indexes are known as auto indexes.

Problem 2: How the Query Optimizer picks indexes

- Part of the Query Optimizer’s job is to determine **if an index can be used to evaluate** a predicate in a query.
- This is basically a comparison between an index key and a constant or variable.

- In addition, the Query Optimizer needs to determine **if the index covers the query**; that is, if the index contains all the columns required by the query.
- It needs to confirm this because, as you’ll hopefully remember, a non-clustered index usually contains only a subset of the columns of the table.

- The Query Optimizer may also consider **using more than one index**, and joining them to cover all the columns required by the query (index intersection).
- If it’s not possible to cover all of the columns required by the query, then the query optimizer **may need to access the base table**, which could be a clustered index, to obtain the remaining columns.

But how can we estimate index cost?

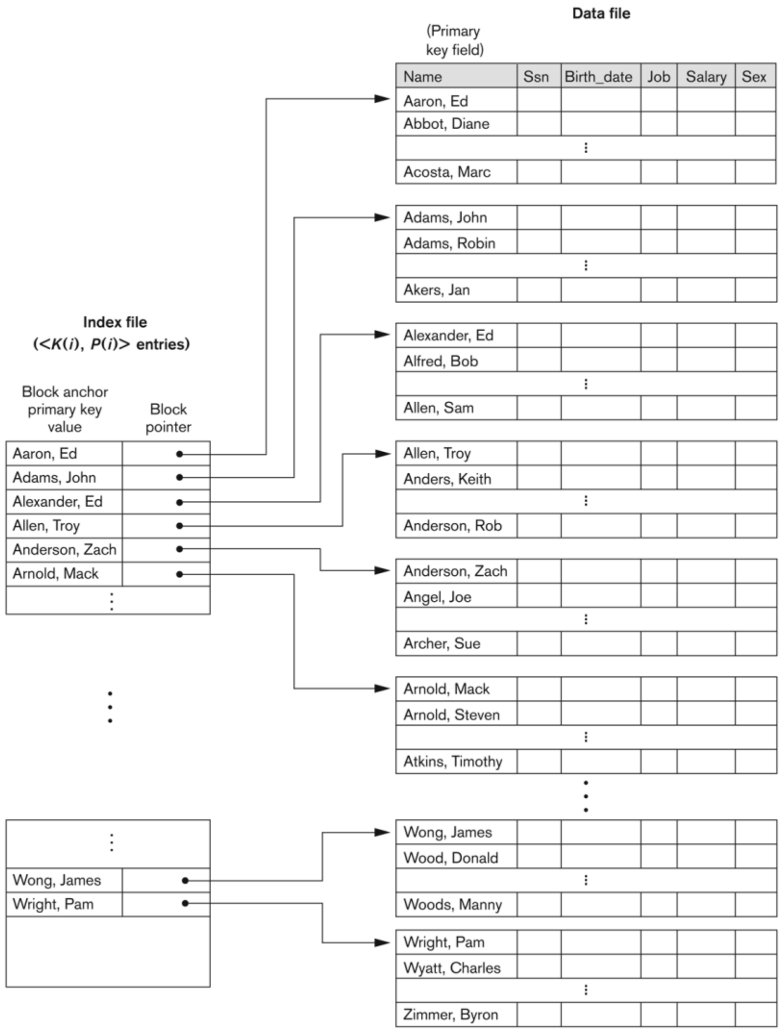
- For both index selection problems,
  we first need an estimate of the ***cost*** of an index lookup

- Need to be able to estimate the costs of different indexes / index types…

We will see this mainly depends on getting estimates of result set size!

Remember …

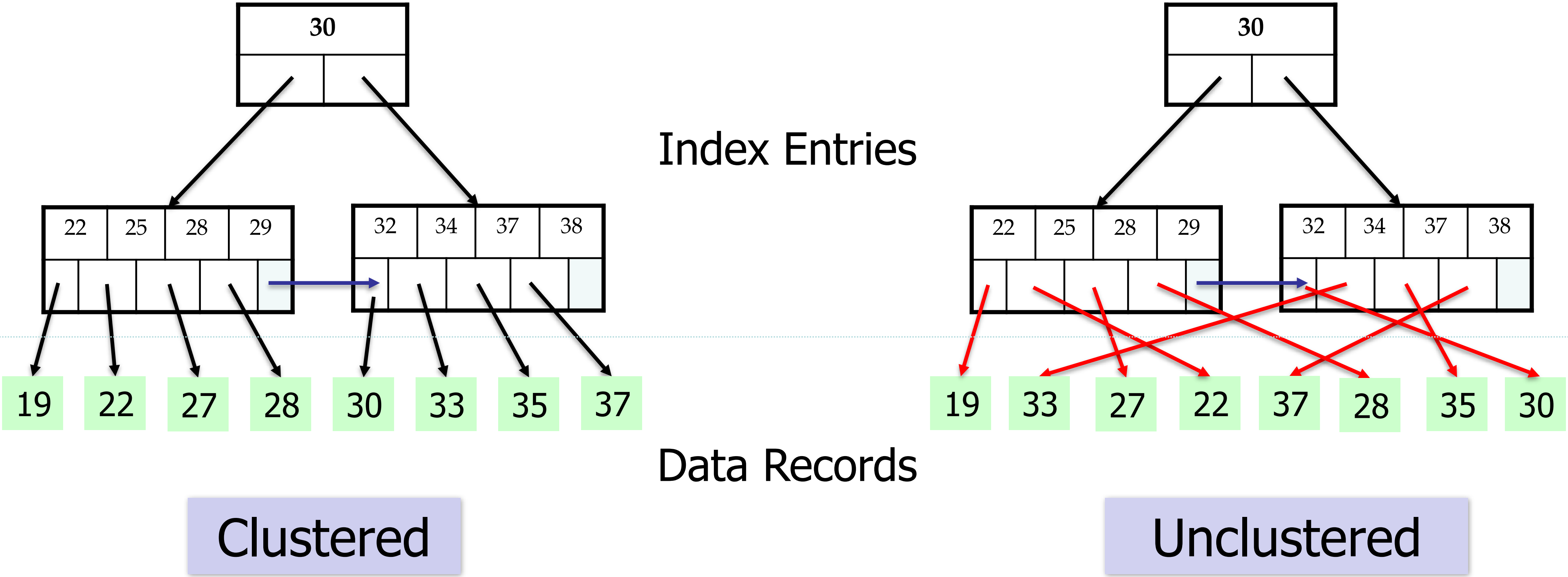
Clustered vs. Unclustered Index



An index is ***clustered*** if the underlying data is ordered in the same way as the index’s data entries.

Remember ...

Clustered vs. Unclustered Index



Index Entries

Data Records

Clustered

Unclustered

Ex: Clustered vs. Unclustered

Let:

f = fanout, which is **in [d+1, 2d+1]**
N = the total number of *pages* we need to index
F = fill-factor (usually ~= 2/3)

Our B+ Tree needs to have room to index ***N / F*** pages!

We need a B+ Tree of height
h = $\left\lceil \log_f \frac{N}{F} \right\rceil$!

Cost to do a range query for M entries over N-page file (P per page):

- Clustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - **To scan: 1 random IO +** $\left\lceil \frac{M-1}{P} \right\rceil$ **sequential IO**

- Unclustered:
  - To traverse: $\text{Log}_f(1.5N)$
  - **To scan: ~ M random IO**

Ex: Clustered vs. Unclustered

Let:

- ***f*** = fanout, which is **in [d+1, 2d+1]**
- ***N*** = the total number of *pages* we need to index
- ***F*** = fill-factor (usually ~= 2/3)

Our B+ Tree needs to have room to index ***N / F*** pages!

We need a B+ Tree of height
h = $\left\lceil \log_f \frac{N}{F} \right\rceil$!

Cost to do a range query for M entries over N-page file (P per page):

- Clustered:
  - To traverse: Log$_f$(1.5N)
  - **To scan: 1 random IO +** $\left\lceil \frac{M-1}{P} \right\rceil$ **sequential IO**
- Unclustered:
  - To traverse: Log$_f$(1.5N)
  - **To scan: ~ M random IO**

How can we have good estimates of M ?

Let's say that:

- Random IO = ~10ms
- Sequential IO = free

~ 1 random IO = 10ms

~ ***M*** random IO = M*10ms

If M = 1, then there is no difference!
If M = 100,000 records, then difference is ~10min. Vs. 10ms!

HISTOGRAMS & IO COST ESTIMATION

IO Cost Estimation via Histograms

- For **index selection**:
  - What is the cost of an index lookup?

- Also for **deciding which algorithm to use**:
  - Ex: To execute $\mathrm{R} \bowtie S$, which join algorithm should DBMS use?
  - **What if we want to compute $\boldsymbol{\sigma_{A>10}(\mathbf{R}) \bowtie \sigma_{B=1}(S)}$?**

- In general, we will need some way to ***estimate intermediate result set sizes***

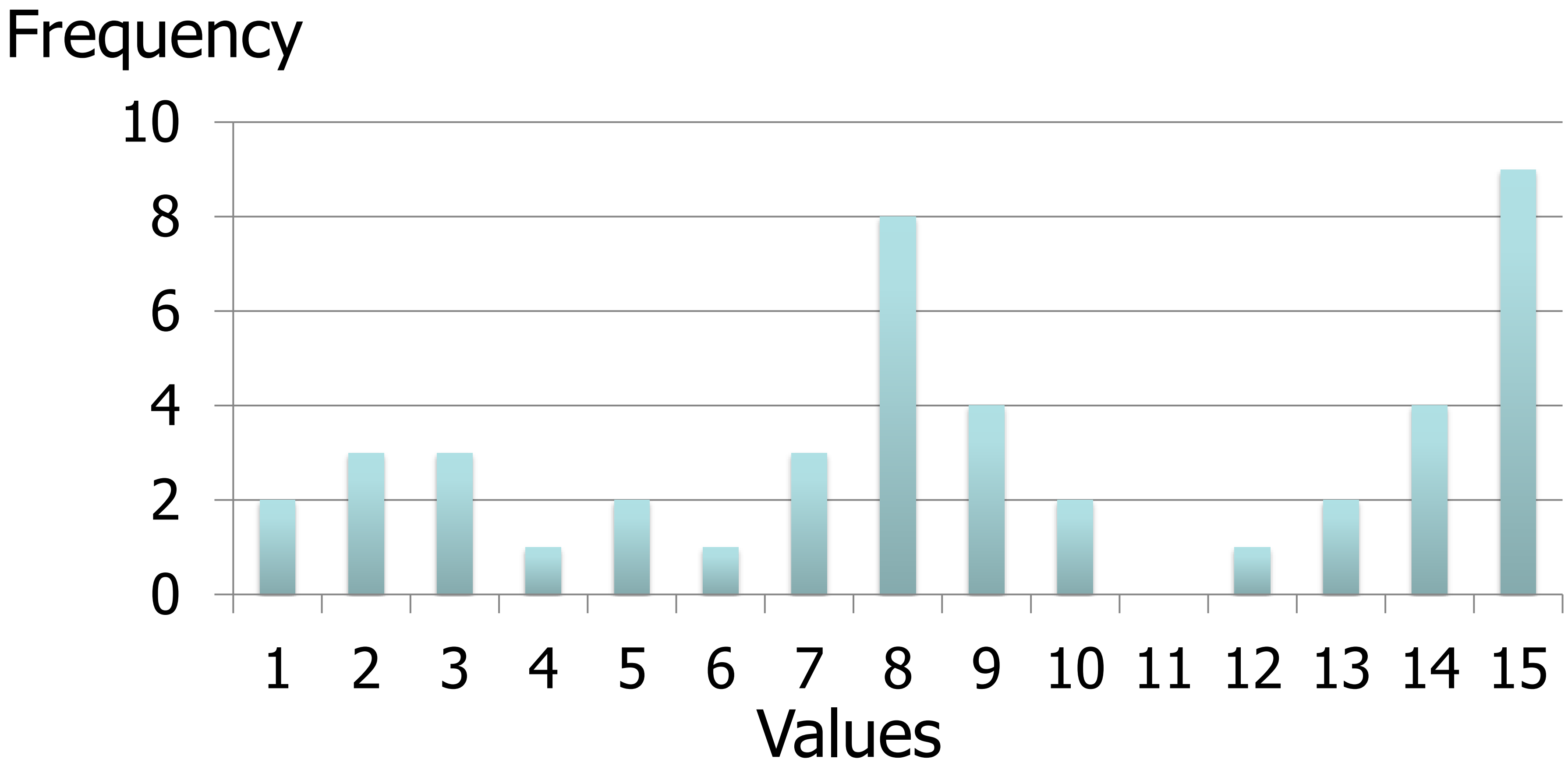
Histograms provide a way to efficiently store estimates of these quantities

Histograms

- A histogram is a set of value ranges (“buckets”) and the frequencies of values in those buckets occurring

- How to choose the buckets?
  - Equiwidth & Equidepth

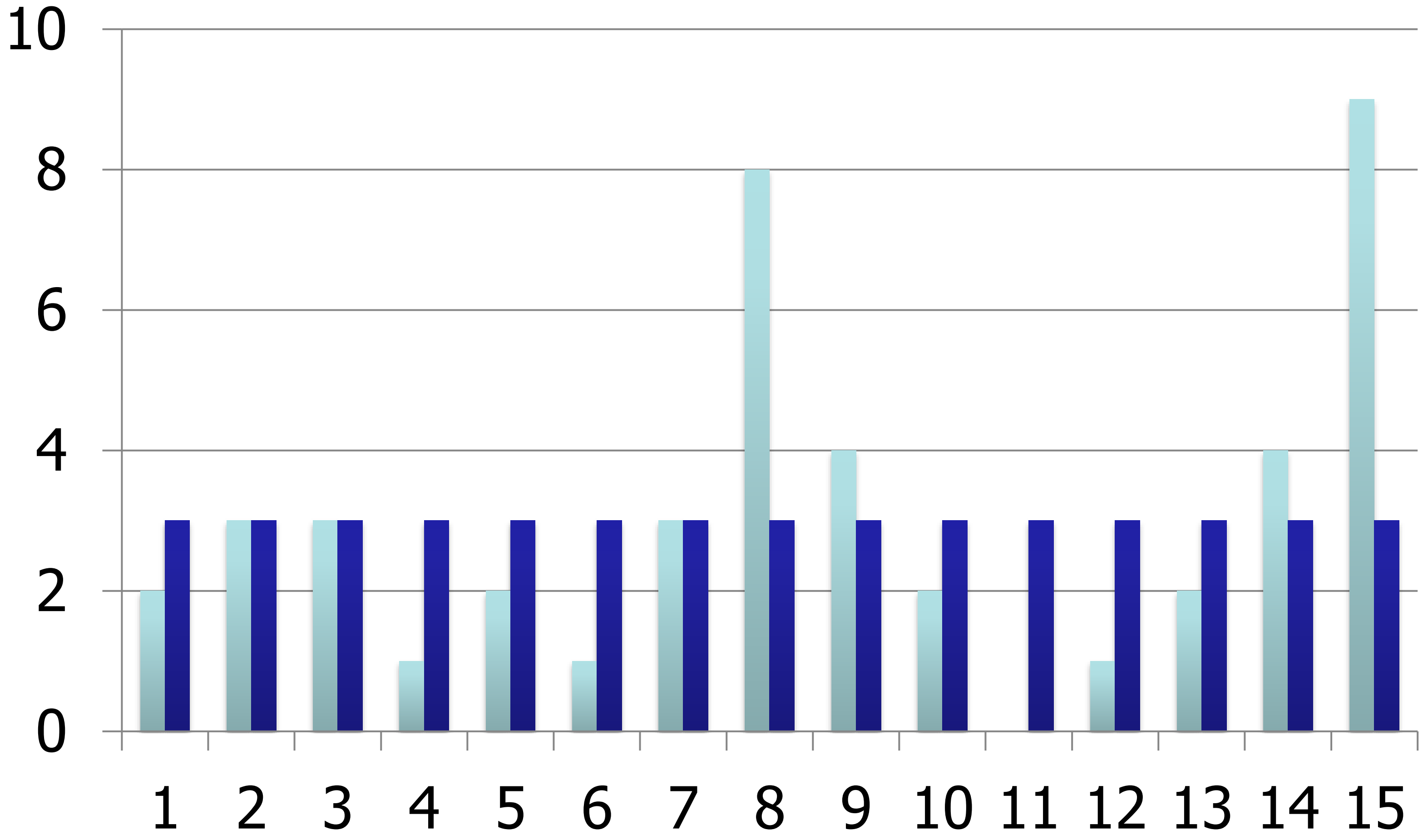
Example



How do we compute how many values between 8 and 10? (Yes, it's obvious)

Problem: counts take up too much space!

Full vs. Uniform Counts



How much space do the full counts (bucket_size=1) take?

How much space do the uniform counts (bucket_size=ALL) take?

Fundamental Tradeoffs

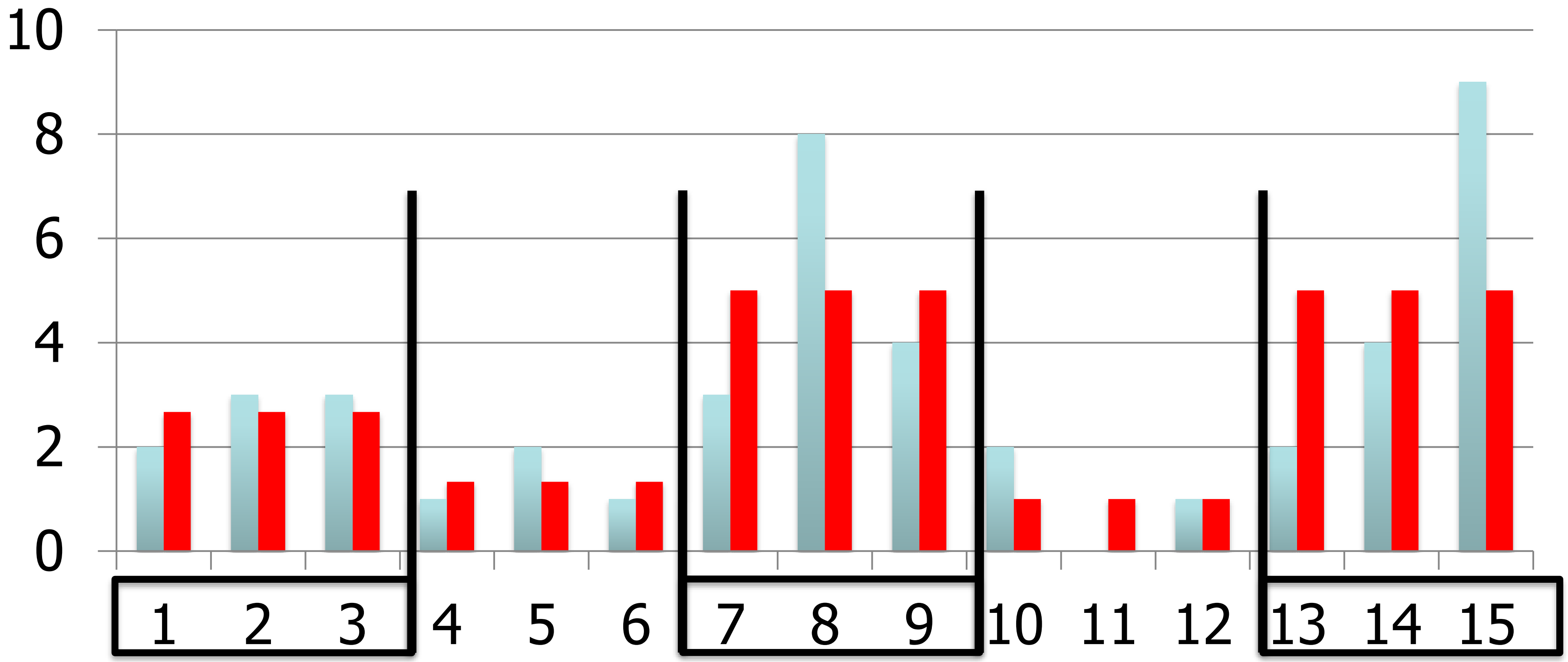
- Want high resolution (like the full counts)

- Want low space (like uniform)

- Histograms are a compromise!

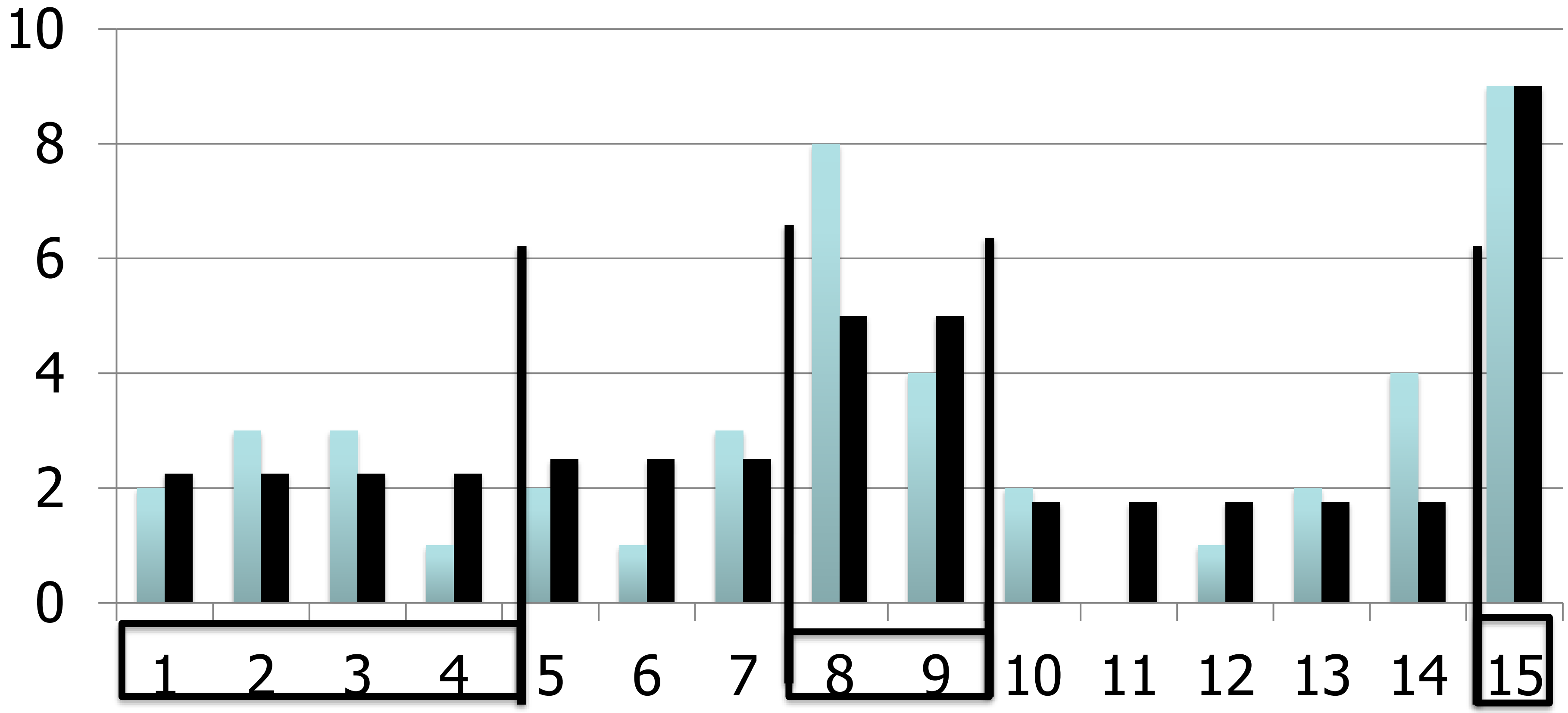
So how do we compute the “bucket” sizes?

Equi-width



All buckets roughly the same width

Equidepth



All buckets contain roughly the same number of items (total frequency)

Histograms

- Simple, intuitive and popular

- Parameters: # of buckets and type

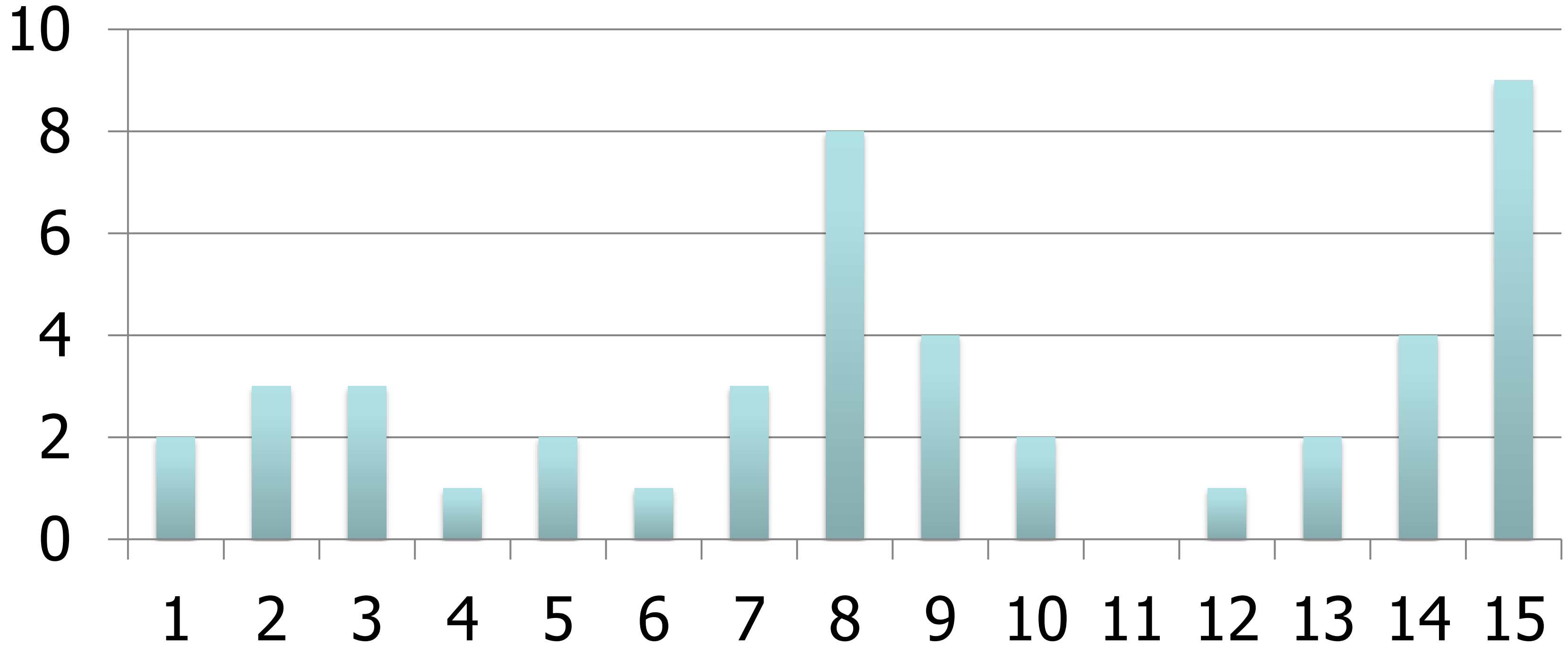
- Can extend to many attributes (multidimensional)

Maintaining Histograms

- Histograms require that we update them!
  - Typically, you must run/schedule a command to update statistics on the database
  - Out of date histograms can be terrible!

- There is research work on self-tuning histograms and the use of query feedback
  - Oracle 11g

Nasty example



1. we insert many tuples with value > 16
2. we do **not** update the histogram
3. we ask for values > 20?

Compressed Histograms

- One popular approach:
  1. Store the most frequent values and their counts explicitly
  2. Keep an equiwidth or equidepth one for the rest of the values

People continue to try all manner of fanciness here

wavelets, graphical models, entropy models,…