



THE UNIVERSITY
of EDINBURGH

Advanced Database Systems

Spring 2026

Lecture #17:

Query Optimisation: Costing

R&G: Chapter 15

WHAT IS NEEDED FOR QUERY OPTIMISATION

Given: A closed set of operators

- Relational operators (table in, table out)

- Physical implementations (of those operators and a few more)

Plan space

- Based on relational equivalences, different implementations

Cost estimation

- Cost formula & size estimation for each physical operator

Search algorithm

- To sift through the plan space and find lowest cost option!

COST ESTIMATION

For each plan considered, must estimate total cost:

Must estimate **cost of each operation** in plan tree

Depends on input cardinalities

Already discussed costs for various operators (sequential scan, index scan, joins, etc.)

Must estimate **size of result** for each operation in tree!

Because it determines downstream input cardinalities!

Use information about the input relations

For selections and joins, assume independence of predicates

In System R, cost boils down to a single number: **#I/Os + CPU-factor * #tuples**

Second term estimate the cost of tuple processing

STATISTICS AND CATALOGS

System catalogs store internal statistics about tables, attributes, and indexes

Typically contain at least:

STATISTIC	MEANING
NTuples	# of tuples in a table (cardinality)
NPages	# of disk pages in a table or index
Low/High	min/max value in a column
NKeys	# of distinct values in a column
Height	the height of an index

Can also keep more detailed statistical information on data values (e.g., histograms)

Catalogs are updated periodically

Users can also manually refresh them (e.g., ANALYZE in PostgreSQL)

Too expensive to do continuously. Lots of approximation anyway, so a little slop is OK

SIZE ESTIMATION AND SELECTIVITY

Max output cardinality = product of input cardinalities

Selectivity (sel) associated with each term

Reflects the impact of the term in reducing result size

Selectivity = $|\text{output}| / |\text{input}|$

Sometimes called “Reduction Factor” (RF)

Always between 0 and 1

Avoid confusion:

“highly selective” in common English is opposite of a high selectivity value
($|\text{output}|/|\text{input}|$ high!)

SELECTION ESTIMATES

The **selectivity** (**sel**) of a predicate **P** is the fraction of tuples that qualify

Equality predicates on unique keys are easy to estimate

```
SELECT * FROM Students
WHERE sid = 123
```

What about more complex predicates?
What is their selectivity?

```
SELECT * FROM Students
WHERE age = 21
```

Formula depends on type of predicate

Equality, range, negation, conjunction, disjunction

```
SELECT * FROM Students
WHERE age > 22
AND dept = 'CS'
```

SELECTIONS - COMPLEX PREDICATES

Assume attribute *age* in relation *Students* has five distinct values (20-24)

$$NKeys(age) = 5$$

Equality predicate

$$sel(A = constant) = 1 / NKeys(A)$$

Example: $sel(age = 22) = 1/5$

```
SELECT * FROM Students
WHERE age = 22
```

```
SELECT * FROM Students
WHERE age > 22
```

Range predicate

$$sel(A > a) = (High(A) - a) / (High(A) - Low(A) + 1) \quad (\text{when } A \text{ is integer-valued column})$$

$$sel(A > a) = (High(A) - a) / (High(A) - Low(A)) \quad (\text{when } A \text{ is floating-valued column})$$

Example: $sel(age > 22) = (24 - 22) / (24 - 20 + 1) = 2/5$

SELECTIONS - COMPLEX PREDICATES

Equality predicate

$$\text{sel}(A = B) = 1 / \max \{ \text{NKeys}(A), \text{NKeys}(B) \} \quad (\text{handy for joins, too})$$

Why MAX?

Assume that A-values and B-values are **independent**

Let there be 2 distinct A-values $\{ v_1, v_2 \}$ and 10 distinct B-values $\{ v_1, \dots, v_{10} \}$

What is the probability of matching values?

$$\begin{aligned} P(A = B) &= \sum_i P(A = v_i, B = v_i) = \sum_i P(A = v_i) \cdot P(B = v_i) \\ &= (1/2 \cdot 1/10) + (1/2 \cdot 1/10) + (0 \cdot 1/10) + \dots \\ &= 1/10 = 1 / \max \{ 2, 10 \} \end{aligned}$$

SELECTIONS - COMPLEX PREDICATES

Negation query

$$\text{sel}(\text{not } P) = 1 - \text{sel}(P)$$

Example: $\text{sel}(\text{age} \neq 22) = 1 - 1/5 = 4/5$

Observation: selectivity \approx probability

```
SELECT * FROM Students
WHERE age != 22
```

Conjunction

$$\text{sel}(P1 \wedge P2) = \text{sel}(P1) \cdot \text{sel}(P2)$$

Assumes that the predicates are independent

```
SELECT * FROM Students
WHERE age = 22
AND name LIKE 'A%'
```

Disjunction

$$\text{sel}(P1 \vee P2) = \text{sel}(P1) + \text{sel}(P2) - \text{sel}(P1 \wedge P2)$$

Assumes that the predicates are independent

```
SELECT * FROM Students
WHERE age = 22
OR name LIKE 'A%'
```

RESULT SIZE ESTIMATION FOR JOINS

How to estimate the size of a join between R and S?

Key-foreign key join

Example: S has a foreign key referencing R

The foreign key constraint guarantees $\pi_A(S) \subseteq \pi_A(R)$, thus $|R \bowtie S| = |S|$

Assumes non-null FK values (e.g., if A is part of a primary key in S); otherwise, $|R \bowtie S| \leq |S|$

RESULT SIZE ESTIMATION FOR JOINS

General case: R join S on A which is not a key for either table

Recall algebraic equivalence: $R \bowtie_p S \equiv \sigma_p(R \times S)$

Hence join selectivity is “just” selectivity $\text{sel}(p)$ over a big input $|R| \cdot |S|$!

Total rows: $\text{sel}(p) \cdot |R| \cdot |S|$

Equi-join on A

Match each R -tuple with S -tuple: $|R \bowtie S| \approx |R| \cdot |S| / \text{NKeys}(S.A)$

Symmetrically, for S : $|R \bowtie S| \approx |S| \cdot |R| / \text{NKeys}(R.A)$

The final estimate is the smaller of the two estimates

Overall: $|R \bowtie S| \approx |R| \cdot |S| \cdot \frac{1}{\max \{ \text{NKeys}(R.A), \text{NKeys}(S.A) \}}$

$\text{sel}(R.A = S.A)$

MISSING STATISTICS? Use DEFAULT VALUES

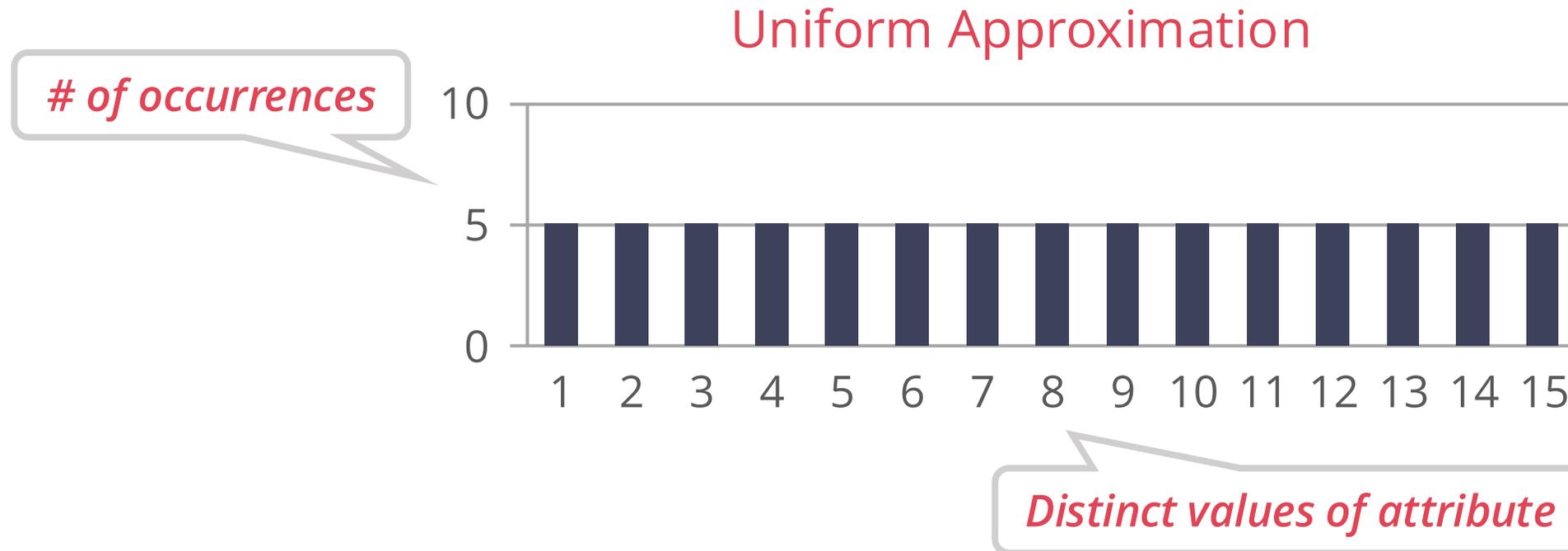
```
/* default selectivity estimate for equalities such as "A = b" */  
#define DEFAULT_EQ_SEL 0.005  
  
/* default selectivity estimate for inequalities such as "A < b" */  
#define DEFAULT_INEQ_SEL 0.3333333333333333  
  
/* default selectivity estimate for range inequalities "A > b AND A < c" */  
#define DEFAULT_RANGE_INEQ_SEL 0.005  
  
/* default selectivity estimate for multirange inequalities "A > b AND A < c" */  
#define DEFAULT_MULTIRANGE_INEQ_SEL 0.005  
  
/* default selectivity estimate for pattern-match operators such as LIKE */  
#define DEFAULT_MATCH_SEL 0.005  
  
/* default selectivity estimate for other matching operators */  
#define DEFAULT_MATCHING_SEL 0.010  
  
/* default number of distinct values in a table */  
#define DEFAULT_NUM_DISTINCT 200
```

Postgres 13.0

src/include/utils/selffuncs.h

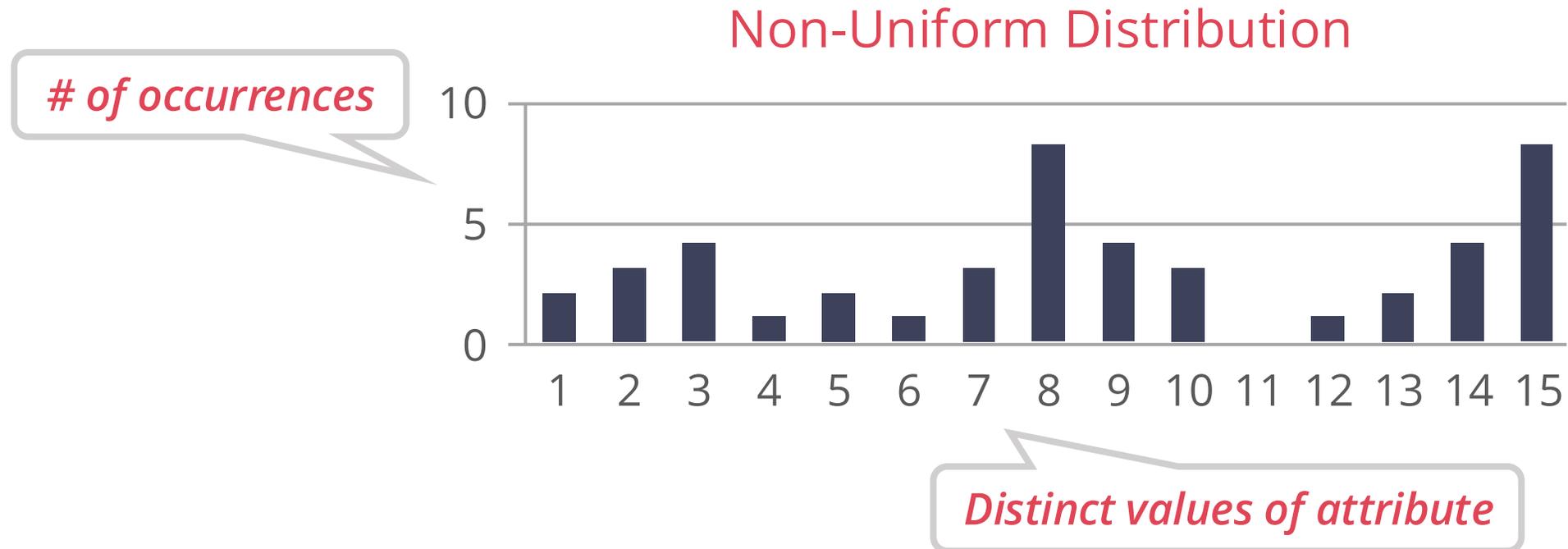
COST ESTIMATION

Our cost formulas assume that data values are uniformly distributed



COST ESTIMATION

In practice, attribute values typically have a non-uniform distribution



HISTOGRAMS

To keep track of this non-uniformity for an attribute A , we can maintain a **histogram** to approximate the actual distribution

Divide the active domain of A into adjacent intervals

Collect statistical parameters for each interval $(b_{i-1}, b_i]$, for example

of tuples r with $b_{i-1} < r.A \leq b_i$

of distinct A values in interval $(b_{i-1}, b_i]$

The histogram intervals are also called **buckets**

TYPES OF HISTOGRAMS

Equi-width histograms

All buckets have the **same width w** or number of distinct values

I.e., boundary $b_{i+1} = b_i + w$ for some fixed width w

Equi-depth histograms

All buckets contain the **same number of tuples** (their width may vary)

Able to adapt to data skew (high uniformity)

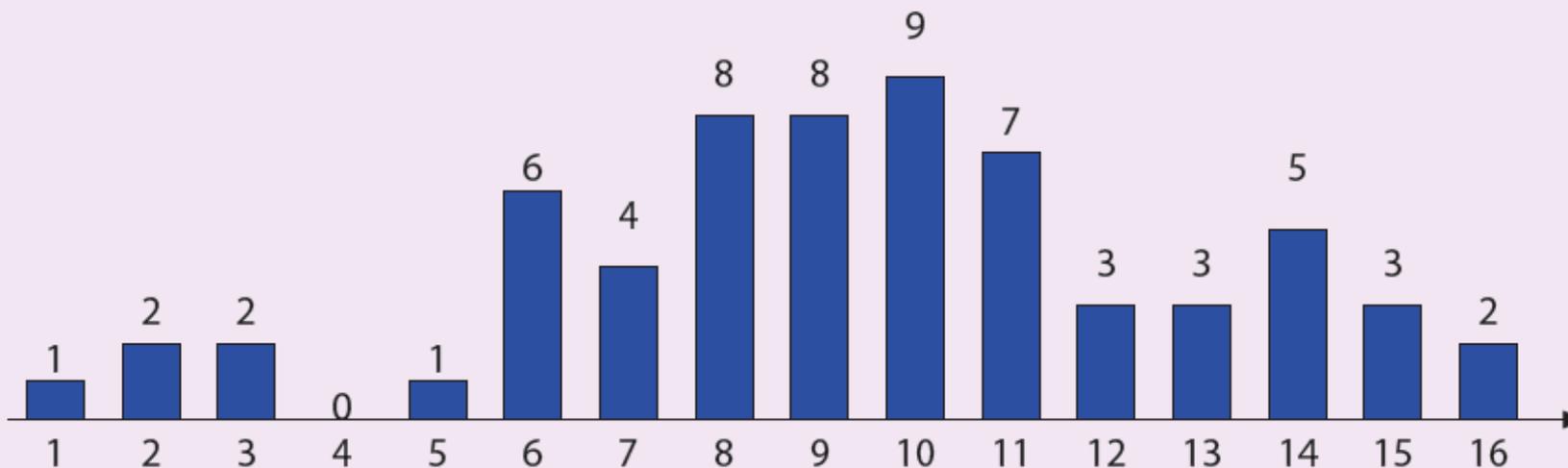
The number of buckets is the tuning knob that defines the trade-off between **histogram resolution** (estimation quality) and **histogram size**

Catalog space is limited!

EQUI-WIDTH HISTOGRAMS

Example (Actual value distribution)

Column A of SQL type INTEGER (domain $\{\dots, -2, -1, 0, 1, 2, \dots\}$).
Actual non-uniform distribution in relation R :



of distinct values = 16, # of tuples = 64

EQUI-WIDTH HISTOGRAMS

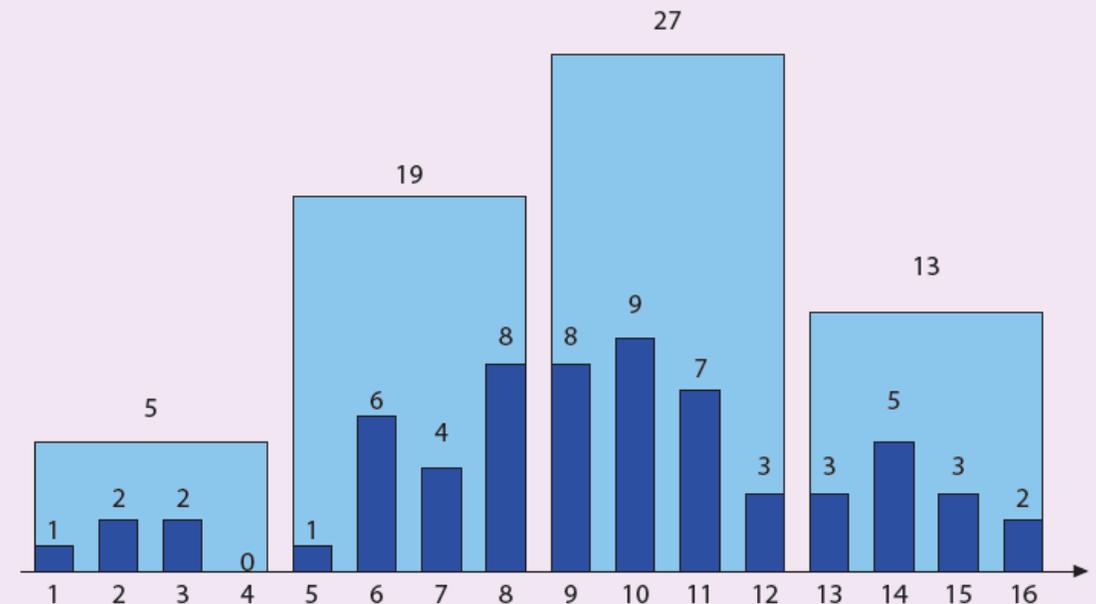
Maintain sum of value frequencies in each bucket
(in addition to bucket boundaries b_i)

Divide active domain of A
into B buckets of equal width

Bucket width w :

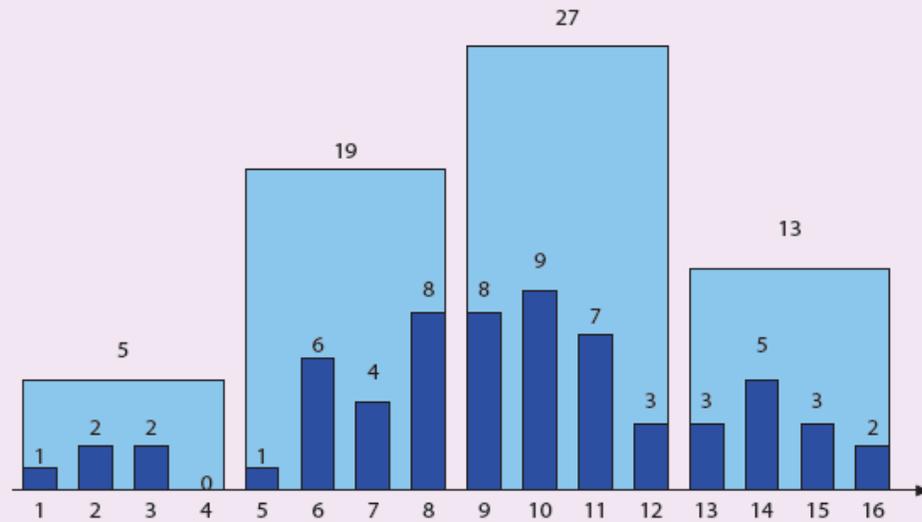
$$w = \frac{\text{High}(A, R) - \text{Low}(A, R) + 1}{B}$$

Example (Equi-width histogram ($B = 4$))



EQUALITY SELECTION

Example ($Q \equiv \sigma_{A=5}(R)$)

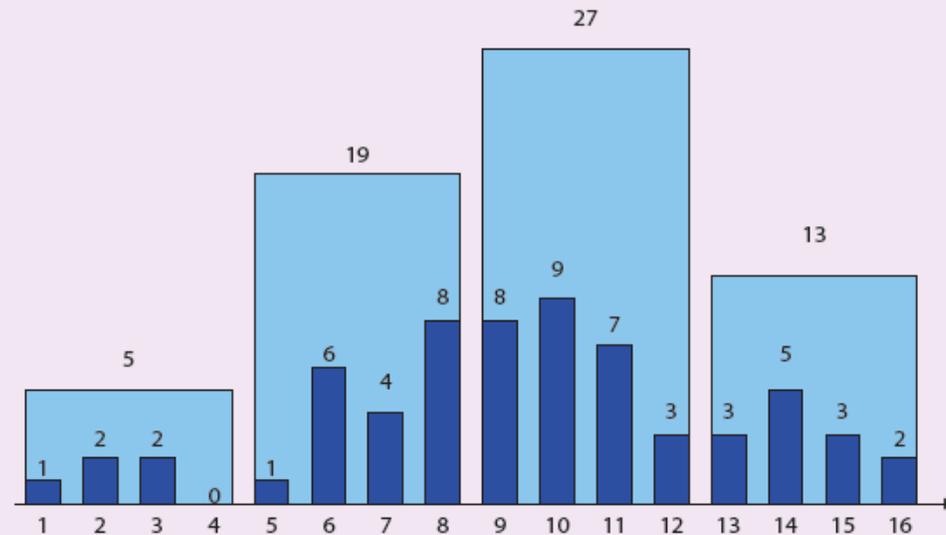


- Value 5 is in bucket $[5, 8]$ (with 19 tuples)
- Assume **uniform distribution within the bucket:**

$$|Q| = 19/w = 19/4 \approx 5 .$$

RANGE SELECTION

Example ($Q \equiv \sigma_{A>7 \text{ AND } A \leq 16}(R)$)



- Query interval $(7, 16]$ covers buckets $[9, 12]$ and $[13, 16]$.
Query interval touches $[5, 18]$.

$$|Q| = 27 + 13 + 19/4 \approx 45 .$$

EQUI-DEPTH HISTOGRAMS

Divide active domain of attribute A into B buckets with *roughly* the same number of tuples in each bucket

Depth d of each bucket will be approximately $|R| / B$

Maintain depth d and bucket boundaries b_i

Intuition:

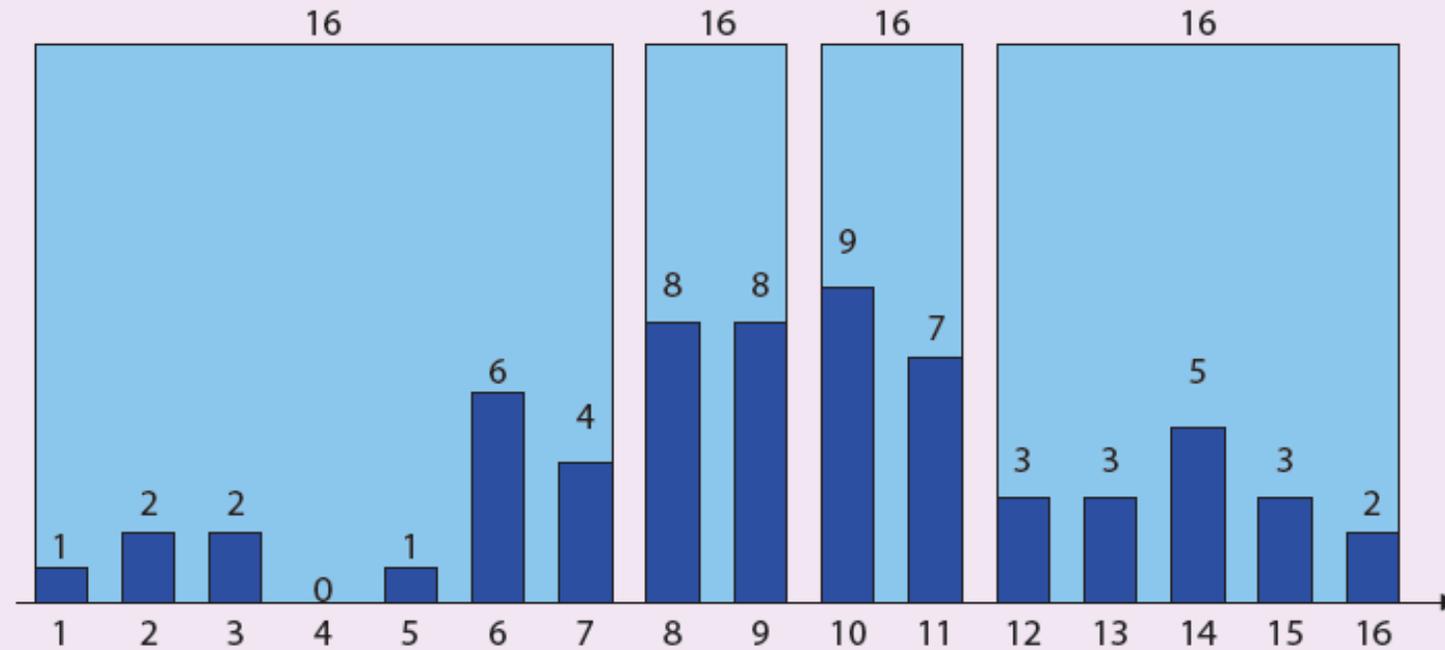
High-value frequencies are more important than low-value frequencies and put in smaller buckets

Equi-depth provides better estimates than equi-width for highly frequent values

Resolution of histogram adapts to skewed value distributions

EQUI-DEPTH HISTOGRAM

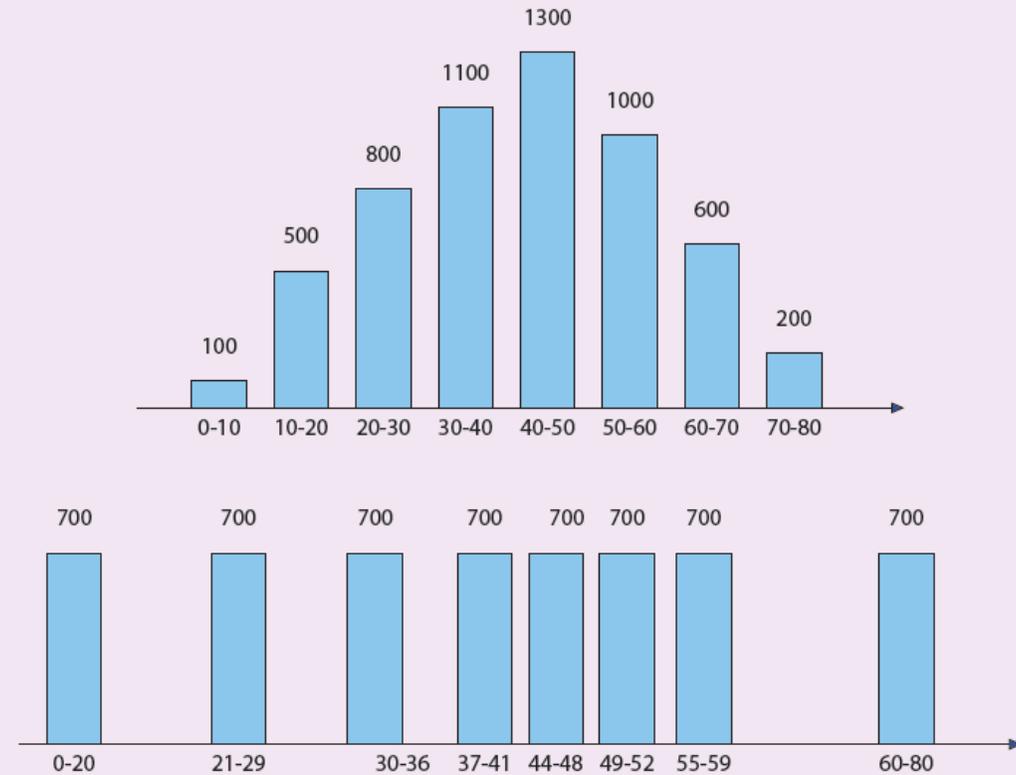
Example (Equi-depth histogram ($B = 4, d = 16$))



COMPARISON

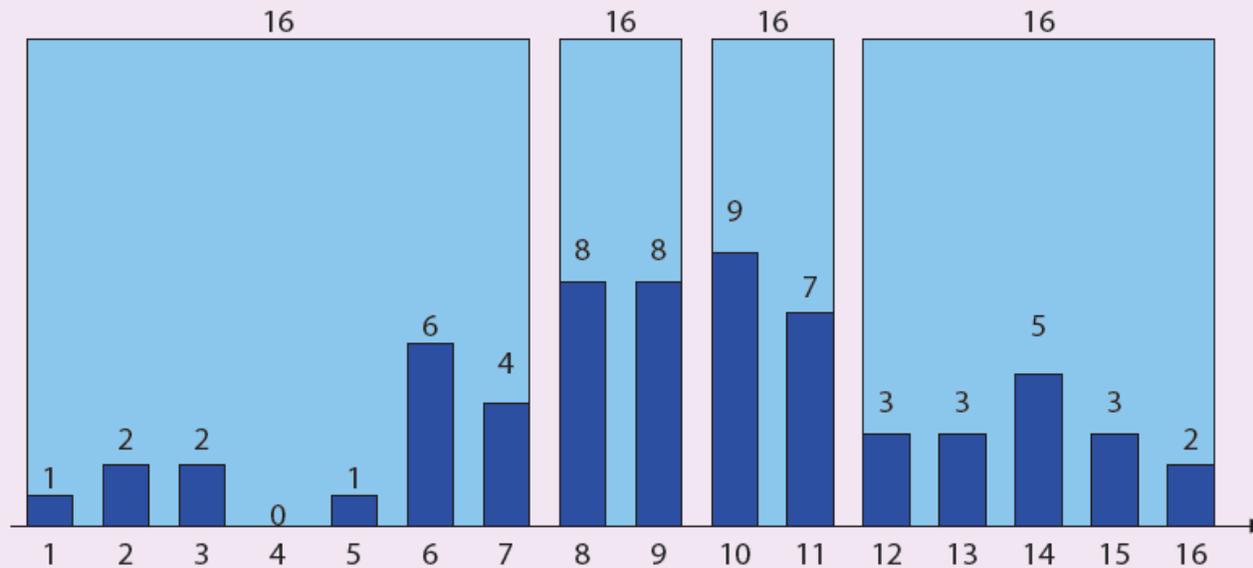
Equi-depth histogram
“invests” bytes in the
densely populated
customer age region
between 30 and 59

Example (Histogram on *customer age* attribute ($B = 8$, $|R| = 5,600$))



EQUALITY SELECTION

Example ($Q \equiv \sigma_{A=5}(R)$)

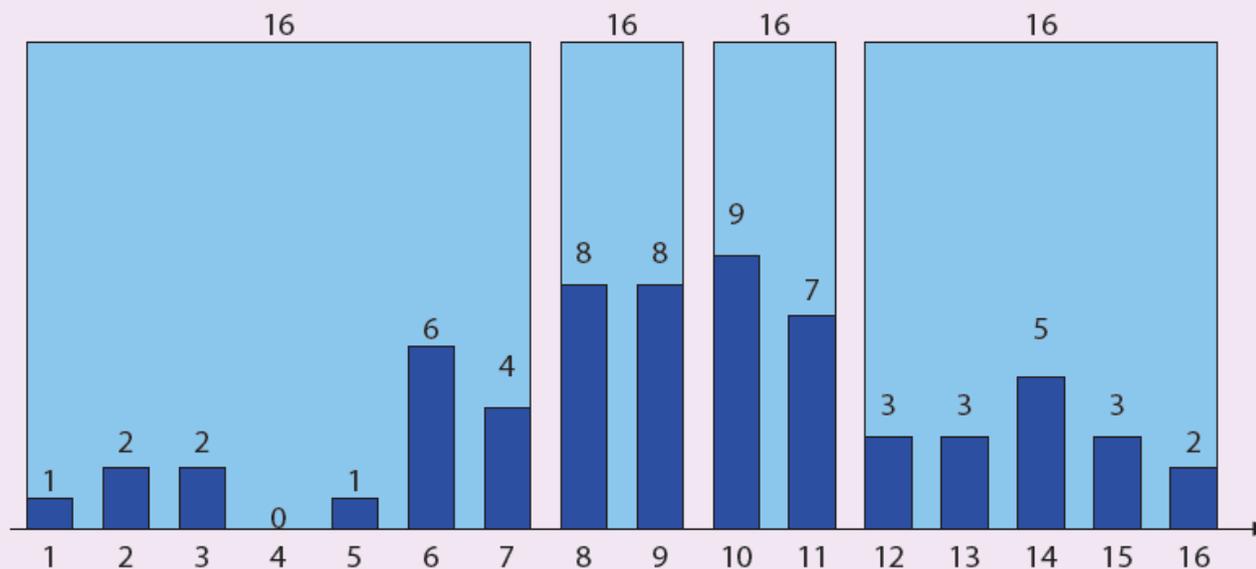


- Value 5 is in first bucket $[1, 7]$ (with $d = 16$ tuples)
- Assume **uniform distribution within the bucket:**

$$|Q| = d/7 = 16/7 \approx 2 .$$

RANGE SELECTION

Example ($Q \equiv \sigma_{A>5 \text{ AND } A \leq 16}(R)$)



- Query interval $(5, 16]$ covers buckets $[8, 9]$, $[10, 11]$ and $[12, 16]$ (all with $d = 16$ tuples). Query interval touches $[1, 7]$.

$$|Q| = 16 + 16 + 16 + 2/7 \cdot 16 \approx 53 .$$

SUMMARY: SELECTIVITY ESTIMATION

We need a way to estimate the size of the intermediate tables

Output size = input size * operator selectivity

Assumption 1: Uniform distribution within histogram bins

Within a bin, fraction of range = fraction of count

Assumption 2: Independent predicates

Selectivity of AND = product of selectivities of predicates

Selectivity of OR = sum of selectivities of predicates – product of selectivities of predicates

Selectivity of NOT = 1 – selectivity of predicates

General joins

Simply compute the selectivity of all predicates

And multiply by the product of the table sizes