THE UNIVERSITY of EDINBURGH

Advanced Databases

Spring 2020

Lecture #10:

Query Optimisation I

Milos Nikolic

Query Optimisation

Remember that SQL is declarative

- User tells the DBMS what answer they want, not how to get the answer

Possibly a big difference in performance based on which plan is used!

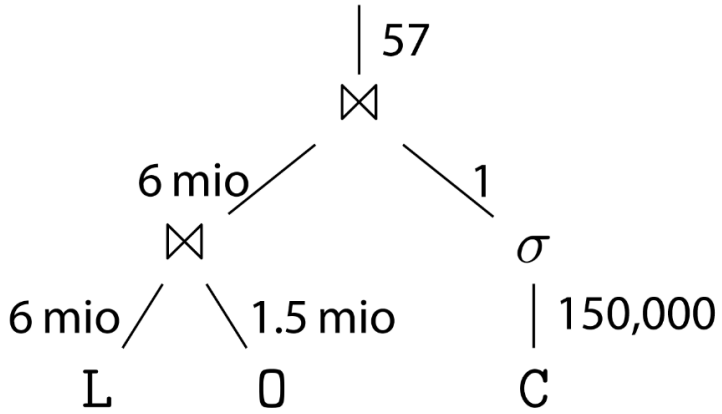
- See last lecture: 1.3 hours vs. 0.45 seconds

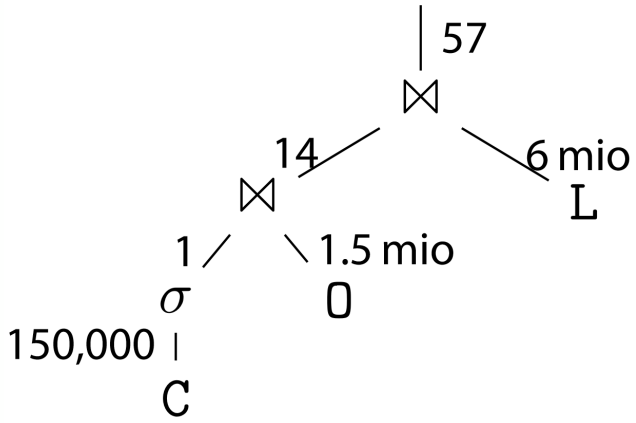
Query Optimisation: Example

Example from TPC-H (standard RDBMS benchmark)

```
SELECT L.partkey, L.quantity, L.extendedprice
  FROM Lineitem L, Orders O, Customer C
 WHERE L.orderkey = O.orderkey
   AND O.custkey = C.custkey
   AND C.name = 'IBM Corp'
```

Two plans, left hand likely orders of magnitude worse:





Query Optimisation: Overview

Heuristics/rule-based rewriting

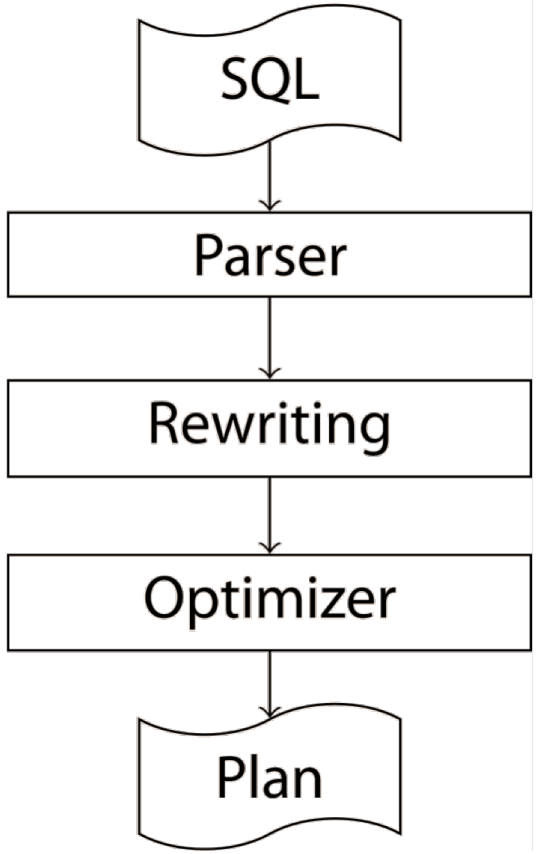
Rewrite the query to remove stupid/inefficient things

Apply equivalence rules of relational algebra

Cost-based search

Use a **cost model** to evaluate multiple equivalent plans and pick the one with the lowest cost

Hard optimisation problem:
typically only approximate solution



SQL Parser

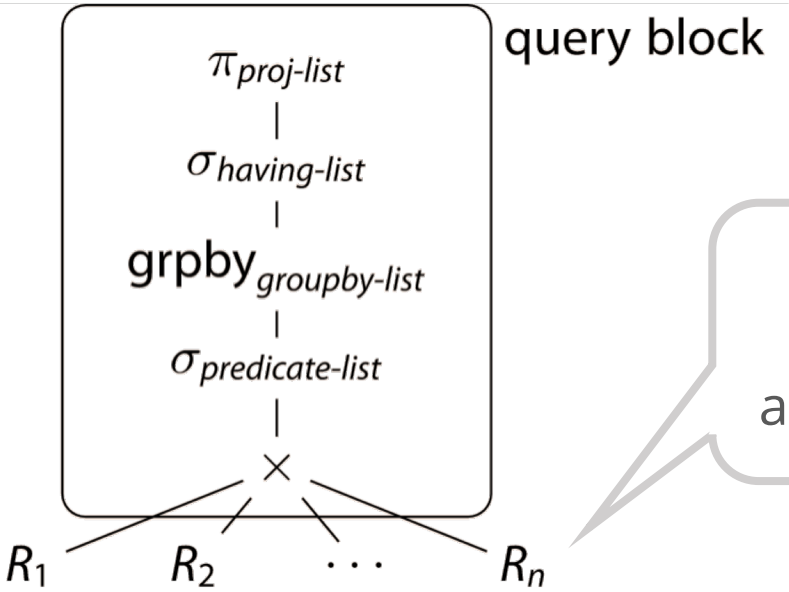
Performs syntactic & semantic analysis

Builds internal representation of the input query

SELECT-FROM-WHERE clauses translated into **query blocks**

```
SELECT  proj-list
  FROM  R1 , R2 , ... , Rn
 WHERE  predicate-list
 GROUP  BY  groupby-list
HAVING  having-list
```

→



Each R_i can be a base relation or another query block

Query Rewriting

Two relational algebra expressions are **equivalent** if they generate the same set of tuples on any database instance

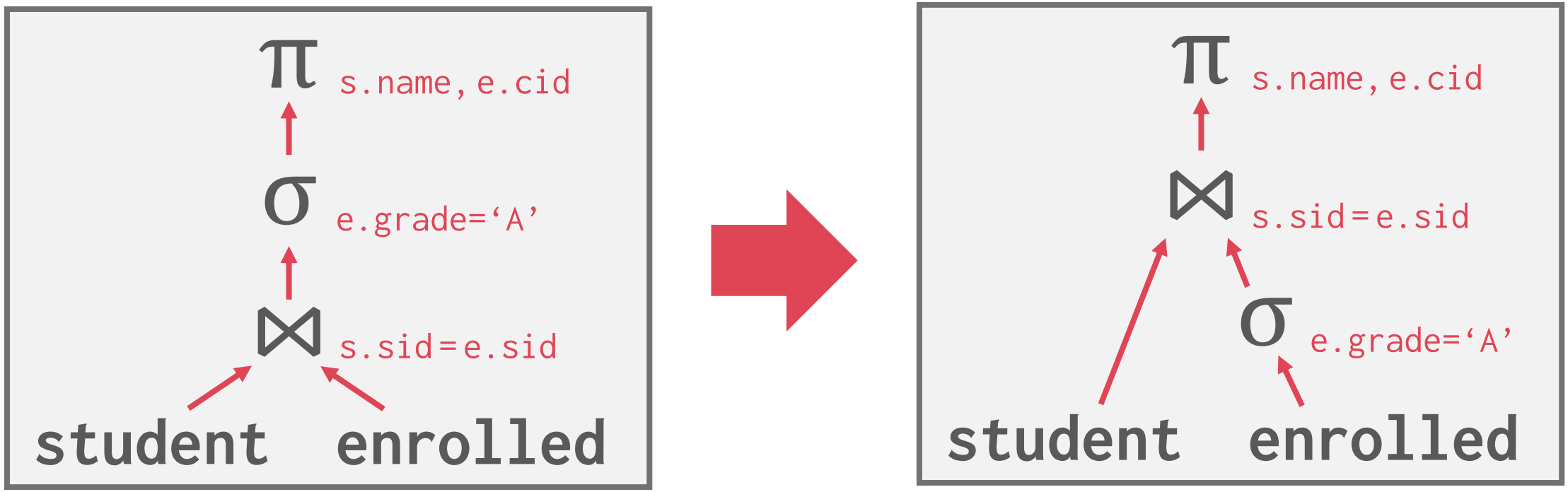
The query rewriter applies heuristics & RA rules, without looking into the actual database state (no info about cardinalities, indices, etc.)

Separated from cost-based optimisation to reduce search space

- Often only a few, very useful rules are applied
- Too expensive to explore all possibilities
- Rule-system often not confluent

Predicate Pushdown

```
SELECT  s.name, e.cid
  FROM  student AS s, enrolled AS e
 WHERE  s.sid = e.sid
   AND  e.grade = 'A'
```



Relational Algebra Equivalences

```
SELECT  s.name, e.cid
  FROM  student AS s, enrolled AS e
 WHERE  s.sid = e.sid
   AND  e.grade = 'A'
```

$$\pi_{\text{name, cid}} (\sigma_{\text{grade='A'}} (\text{student} \bowtie \text{enrolled}))$$

$$\equiv$$

$$\pi_{\text{name, cid}} (\text{student} \bowtie (\sigma_{\text{grade='A'}} (\text{enrolled}))$$

Relational Algebra Equivalences

Selections

Filter as early as possible

Reorder predicates so that the DBMS applies the most selective one first

Break complex predicates and push down

$\sigma_{p1 \wedge p2 \wedge \ldots \wedge pn}(R) = \sigma_{p1}(\sigma_{p2}(\ldots \sigma_{pn}(R)))$

Simplify complex predicates

(X = Y AND Y = 3) ⟶ X = 3 AND Y = 3

L.TAX * 100 < 5 ⟶ L.TAX < 0.05

Relational Algebra Equivalences

Projections

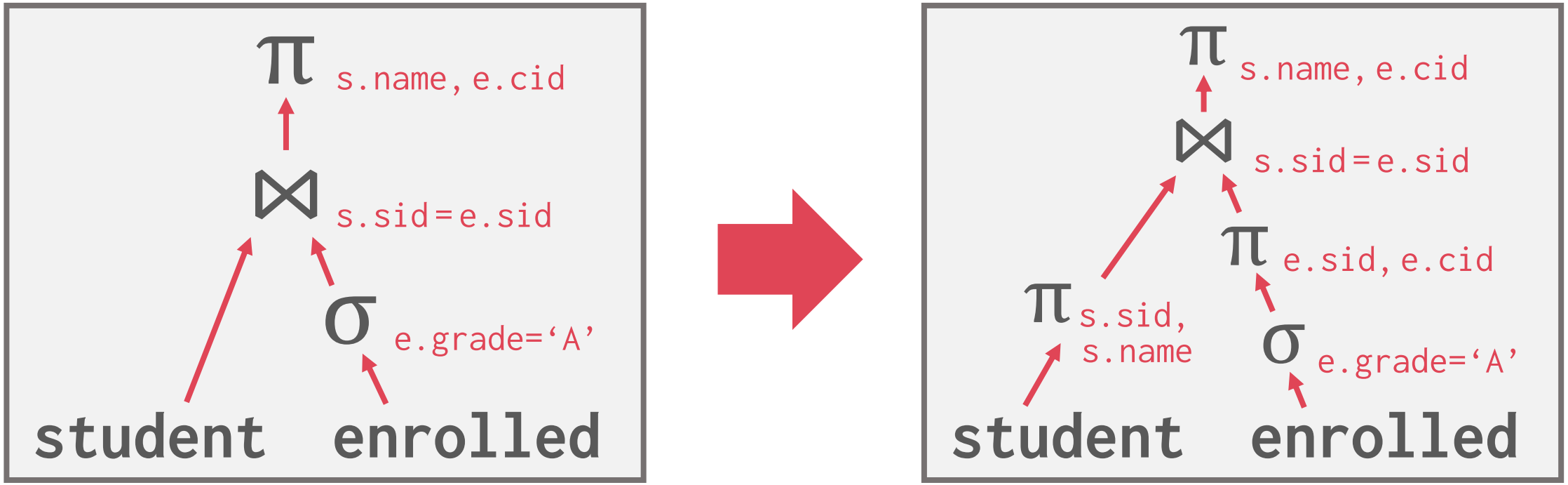
Perform them early to create smaller tuples and reduce intermediate results (if duplicates are eliminated)

Project out all attributes except the ones requested or required (e.g., joining keys)

This is not important for a column store…

Projection Pushdown

```
SELECT  s.name, e.cid
  FROM  student AS s, enrolled AS e
 WHERE  s.sid = e.sid
   AND  e.grade = 'A'
```



Query Rewriting: Rules I

Commutativity and **associativity** of binary operators

$$R \bowtie S = S \bowtie R \qquad R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

$$R \cup S = S \cup R \qquad R \cup (S \cup T) = (R \cup S) \cup T$$

$$R \cap S = S \cap R \qquad R \cap (S \cap T) = (R \cap S) \cap T$$

$$R \times S = S \times R \qquad R \times (S \times T) = (R \times S) \times T$$

Nested selection or projection

$$\sigma_p(\sigma_{p'}(R)) = \sigma_{p'}(\sigma_p(R)) \qquad \pi_A(\pi_{A'}(R)) = \pi_A(R) \text{ for } A \subseteq A'$$

Query Rewriting: Rules II

Compound **selection** (replace conjunction with nesting):

$$\sigma_{p_1 \wedge \ldots \wedge p_n}(R) = \sigma_{p_1}(\ldots(\sigma_{p_n}(R)))$$

Join creation (similar for other join conditions)

$$\sigma_{R.A=S.B}(R \times S) = R \bowtie_{R.A=S.B} S$$

Usual Boolean rules for logical expressions in selections

$$p_1 \wedge p_2 = p_2 \wedge p_1 \quad p_1 \vee (p_2 \wedge p_3) = (p_1 \vee p_2) \wedge (p_1 \vee p_3) \quad \neg(p_1 \wedge p_2) = \neg p_1 \vee \neg p_2$$

Query Rewriting: Rules III

Swapping operators

OPERATORS	EQUIVALENCE	CONDITION
$\sigma \leftrightharpoons \pi$	$\pi_A\left(\sigma_p(R)\right) = \sigma_p\left(\pi_A(R)\right)$	$\text{attr}(p) \subseteq A$
$\sigma \leftrightharpoons \bowtie,\times$	$\sigma_p(R \bowtie S) = \sigma_p(R) \bowtie S$ $\sigma_p(R \times S) = \sigma_p(R) \times S$	$\text{attr}(p) \subseteq \text{attr}(R)$
$\pi \leftrightharpoons \bowtie$	$\pi_A\left(R \bowtie_p S\right) = \pi_A\left(\pi_{A_R}(R) \bowtie_p \pi_{A_S}(S)\right)$	$A_R = \text{attr}(R) \cap \left(A \cup \text{attr}(p)\right)$ $A_S = \text{attr}(S) \cap \left(A \cup \text{attr}(p)\right)$
$\sigma \leftrightharpoons \cap,\cup$	$\sigma_p(R \circ S) = \sigma_p(R) \circ \sigma_p(S)$	$\circ \in \{\cap,\cup\}$
$\pi \leftrightharpoons \cup$	$\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)$	

Query Rewriting

Many more rewriting rules

- e.g., for **NOT EXISTS**, **ALL**, **ANY**, **GROUP BY** aggregates
- based on laws for quantifiers in relational calculus and algebraic laws

Rewritten query plans often have selections “fixed” to a join or pushed as close as possible to an input relation

- Typical form: $\sigma(R_1) \bowtie \sigma(R_2) \bowtie \ldots$
- But pushing selections down might not always be desirable. Examples?

Best plan comes down to **join ordering algorithms**

More Examples

```
CREATE TABLE R (
  id  INT PRIMARY KEY,
  val INT NOT NULL );
```

Impossible / unnecessary predicates

```
SELECT * FROM R WHERE 1 = 0;
```

empty result

```
SELECT * FROM R WHERE 1 = 1;
```

```
SELECT * FROM R;
```

Join elimination

```
SELECT R1.*
  FROM R AS R1 JOIN R AS R2
    ON R1.id = R2.id;
```

```
SELECT * FROM R;
```

More Examples

```
CREATE TABLE R (
  id  INT PRIMARY KEY,
  val INT NOT NULL );
```

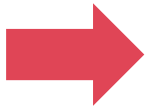
Ignoring nested subquery

```
SELECT * FROM R AS R1
 WHERE EXISTS (SELECT * FROM R AS R2
                 WHERE R1.id = R2.id);
```

```
SELECT * FROM R;
```

Merging predicates

```
SELECT * FROM R
 WHERE val BETWEEN 1 AND 100
    OR val BETWEEN 50 AND 150;
```



```
SELECT * FROM R
 WHERE val BETWEEN 1 AND 150;
```

Introducing Additional Join Conditions

Implicit join through transitivity

```
SELECT * FROM R, S, T
 WHERE R.a = S.b AND S.b = T.c
```

can be turned into

```
SELECT * FROM R, S, T
 WHERE R.a = S.b AND S.b = T.c AND R.a = T.c
```

making the join ordering **(R ⋈ T) ⋈ S** possible (avoids a Cartesian product)

Cost Estimation

Cost-based **optimisation** requires

- cost formula for each operator
- but this depends on input size and thus output of previous sub-query

Selectivity and **cardinality** estimation

- estimated cardinality of a (sub-)query crucial
- cardinality typically measured in pages or rows
- cardinality estimates also valuable when it comes to buffering

Uniformity and **independence** assumption:

- All values of an attribute have the same probability
- Values of different attributes are independent of each other

STATISTICS

The DBMS stores internal statistics about tables, attributes, and indexes in the **system catalog**

The DBMS periodically updates these statistics

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

For each relation R, assume the DBMS maintains the following info:

- |R|: number of tuples in R
- V(A,R): number of distinct values for attribute A

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 people
 WHERE id = 123
```

What about more complex predicates? What is their selectivity?

```
SELECT * FROM people
 WHERE age = 2
```

Formula depends on type of predicate

- Equality, range, negation, conjunction, disjunction

```
SELECT * FROM people
 WHERE age >= 2
   AND status = 'Emp'
```

Selections - Complex Predicates

Assume attribute *age* in relation *people* has five distinct values (0-4)

V(age, people) = 5

Equality predicate: A = constant

sel(A = constant) = 1 / V(A,R)

Example: **sel(age = 2) = 1/5**

```
SELECT * FROM people
 WHERE age = 2
```

Range predicate:

$sel(A > a) = (A_{max} - a) / (A_{max} - A_{min})$

Examples:

sel(age > 2) = (4 – 2) / (4 – 0) = 1/2 (*age* treated as continuous)

sel(age > 2) = sel(age = 3) + sel(age = 4) = 2/5 (*age* treated as categorical)

```
SELECT * FROM people
 WHERE age > 2
```

Selections - Complex Predicates

Negation query

sel(not P) = 1 – sel(P)

Example: sel(age != 2) = 1 – 1/5 = 4/5

Observation: selectivity ≈ probability

```
SELECT * FROM people
 WHERE age != 2
```

Conjunction

sel(P1 ∧ P2) = sel(P1) · sel(P2)

Assumes that the predicates are independent

```
SELECT * FROM people
 WHERE age = 2
   AND name LIKE 'A%'
```

Disjunction

sel(P1 v P2) = sel(P1) + sel(P2) – sel(P1 ∧ P2)

Assumes that the predicates are independent

```
SELECT * FROM people
 WHERE age = 2
    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

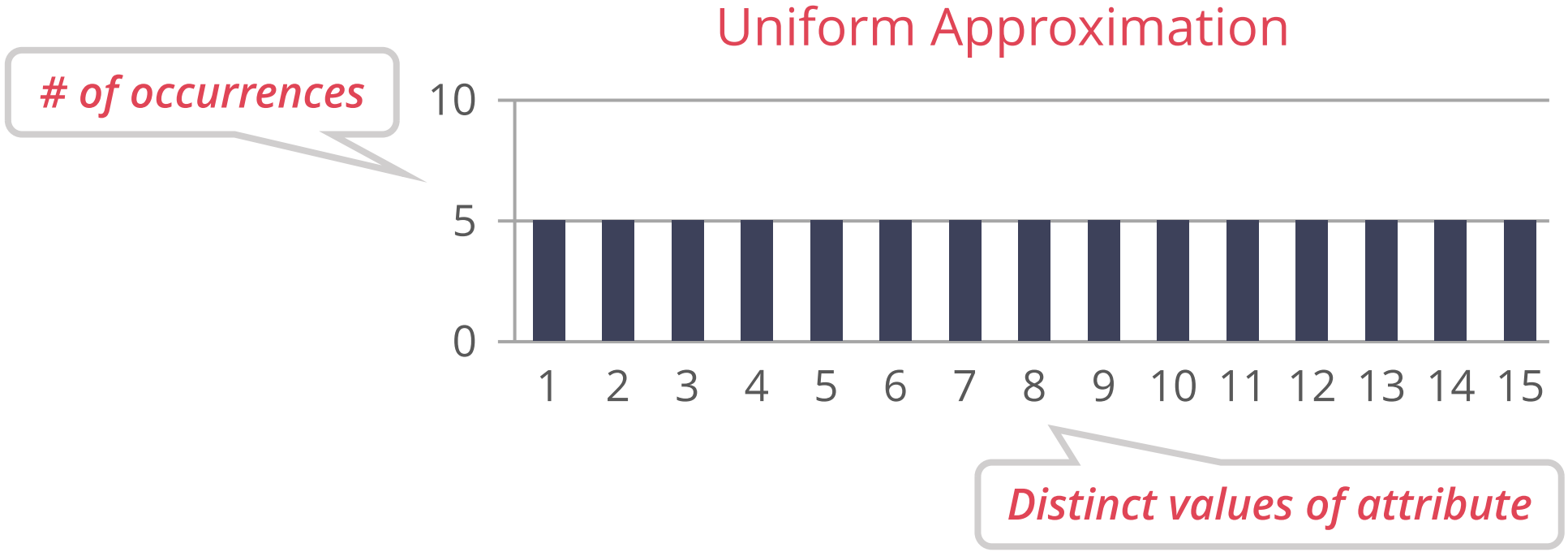
- Example: S has a foreign key referencing R
- The foreign key constraint guarantees $\pi_A(S) \subseteq \pi_A(R)$, thus $|R \bowtie S| = |S|$

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

- Match each R-tuple with S-tuple: $|R \bowtie S| \approx |R| \cdot |S| \,/\, V(A,S)$
- Symmetrically, for S: $|R \bowtie S| \approx |S| \cdot |R| \,/\, V(A,R)$
- **Overall:** $|R \bowtie S| \approx |R| \cdot |S| \,/\, \max\{ V(A,S), V(A,R) \}$

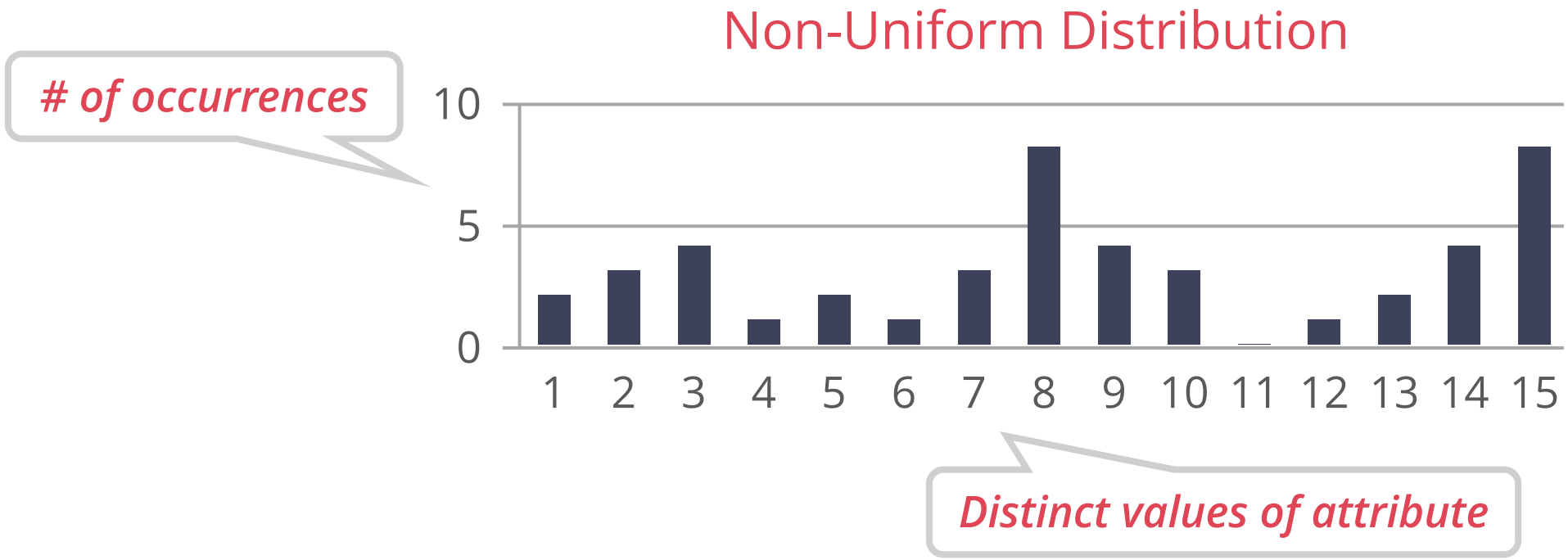
Cost Estimation

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 $\mathbf{b_{i+1} = b_i + w}$ 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)

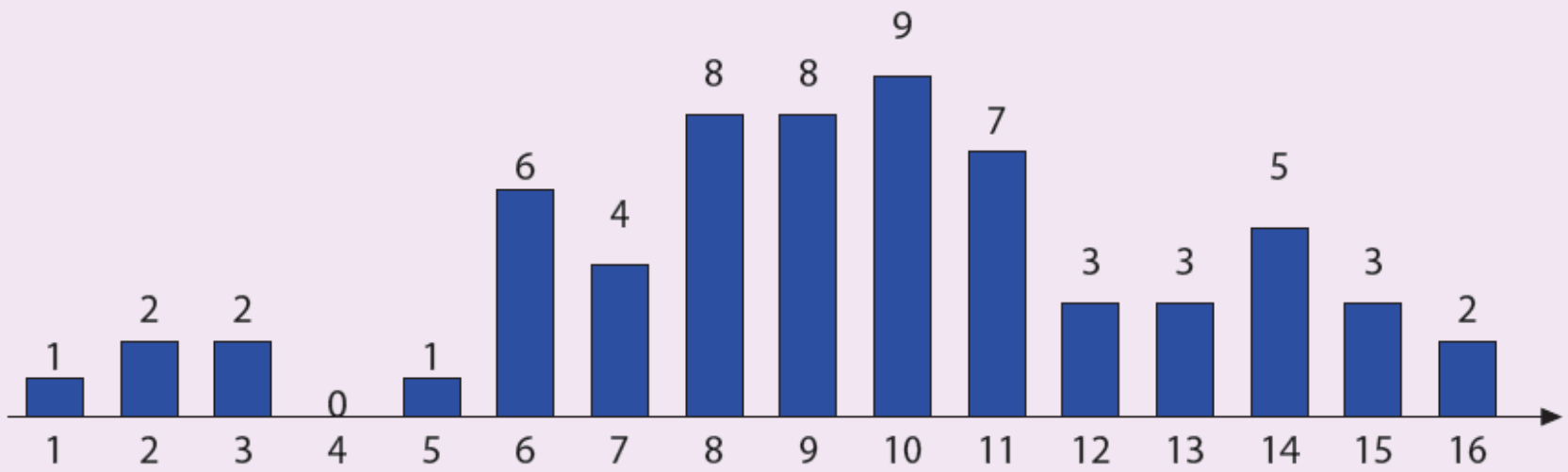
The number of buckets is the tuning knob that defines the trade-off between **histogram resolution** (estimation quality) and **histogram size**

Catalog space is limited

Equi-Width Histograms

Example (Actual value distribution)

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



# of distinct values = 16, # of tuples = 64

Equi-Width Histograms

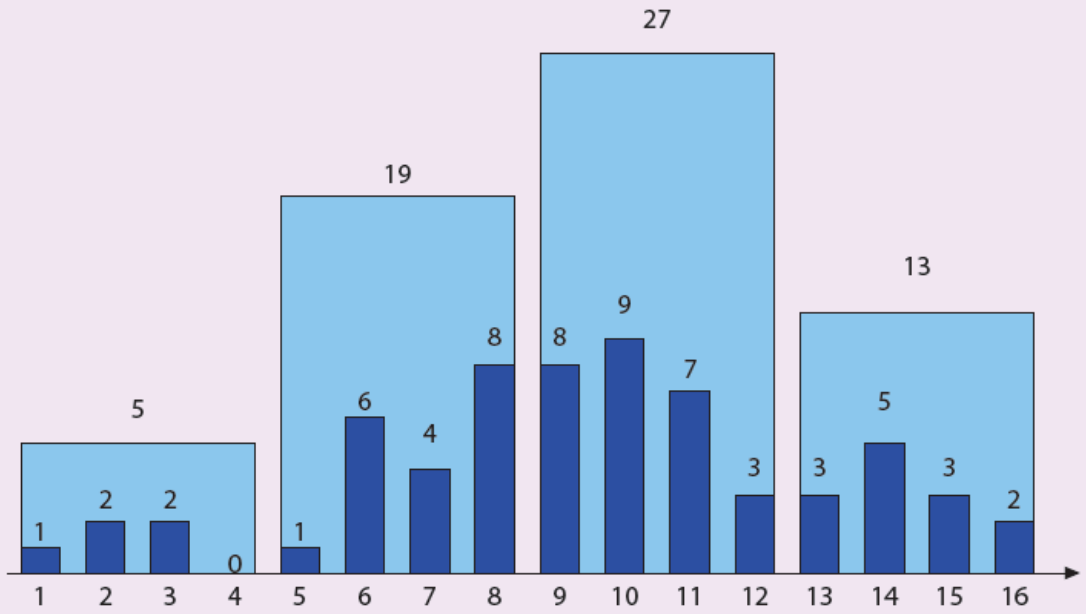
Maintain sum of value frequencies in each bucket
(in addition to bucket boundaries b_i)

Divide active domain of A
into **B** buckets of equal width

Bucket width w:

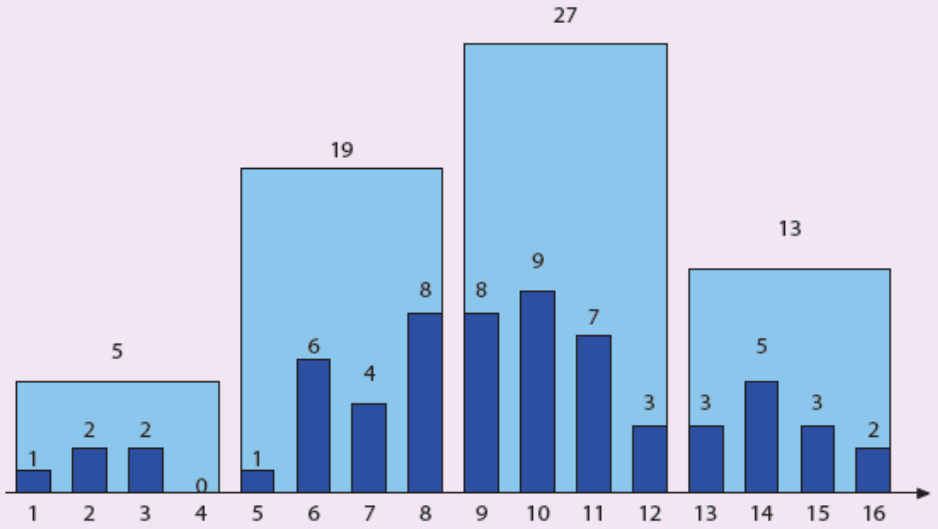
$$w = \frac{High(A, R) - Low(A, R) + 1}{B}$$

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



Equality Selection

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

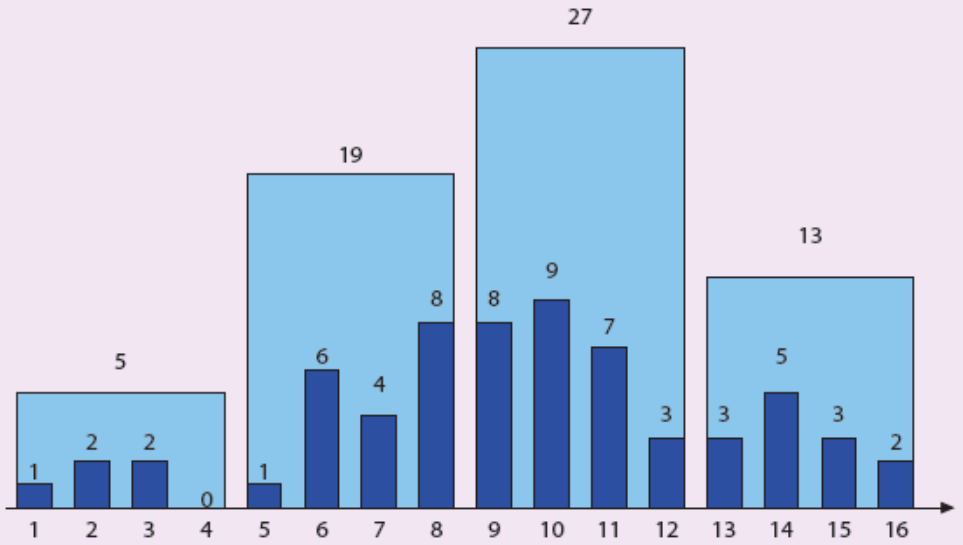


- Value 5 is in bucket $[5, 8]$ (with 19 tuples)
- Assume **uniform distribution within the bucket**:

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

Range Selection

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



- 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

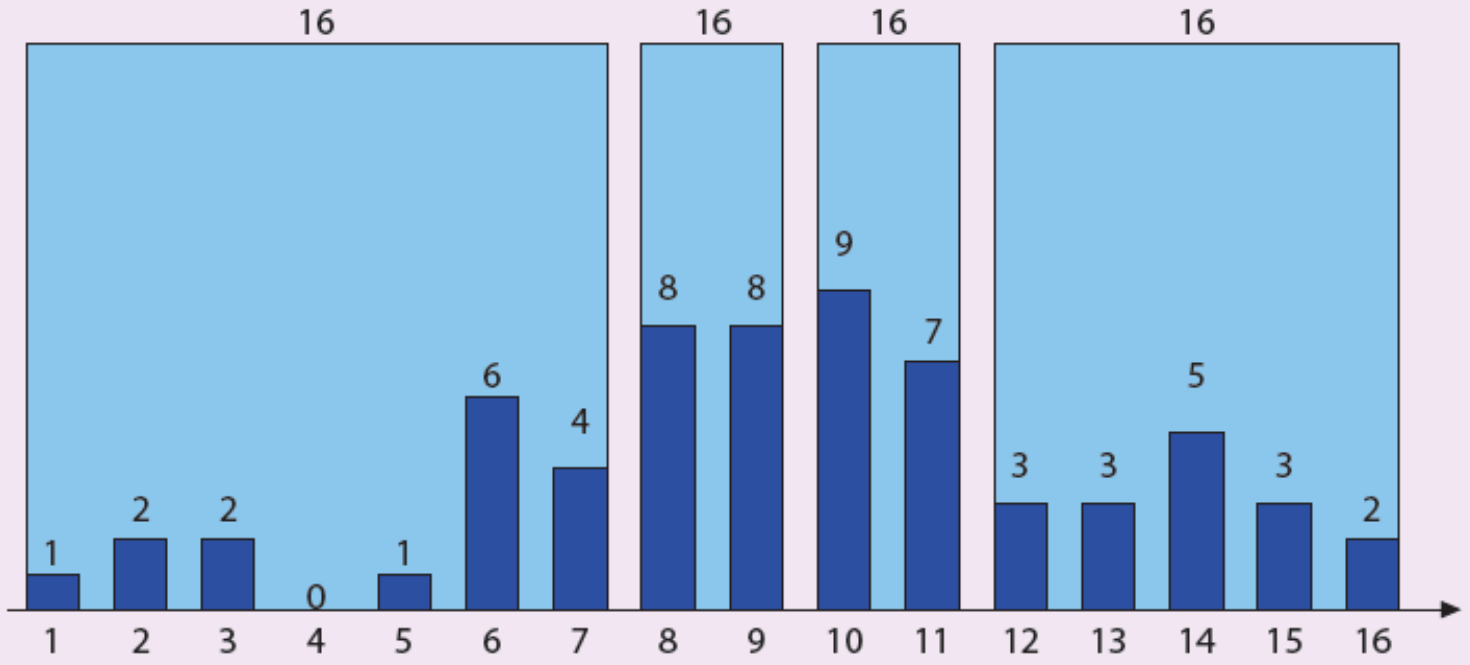
Intuition:

- 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

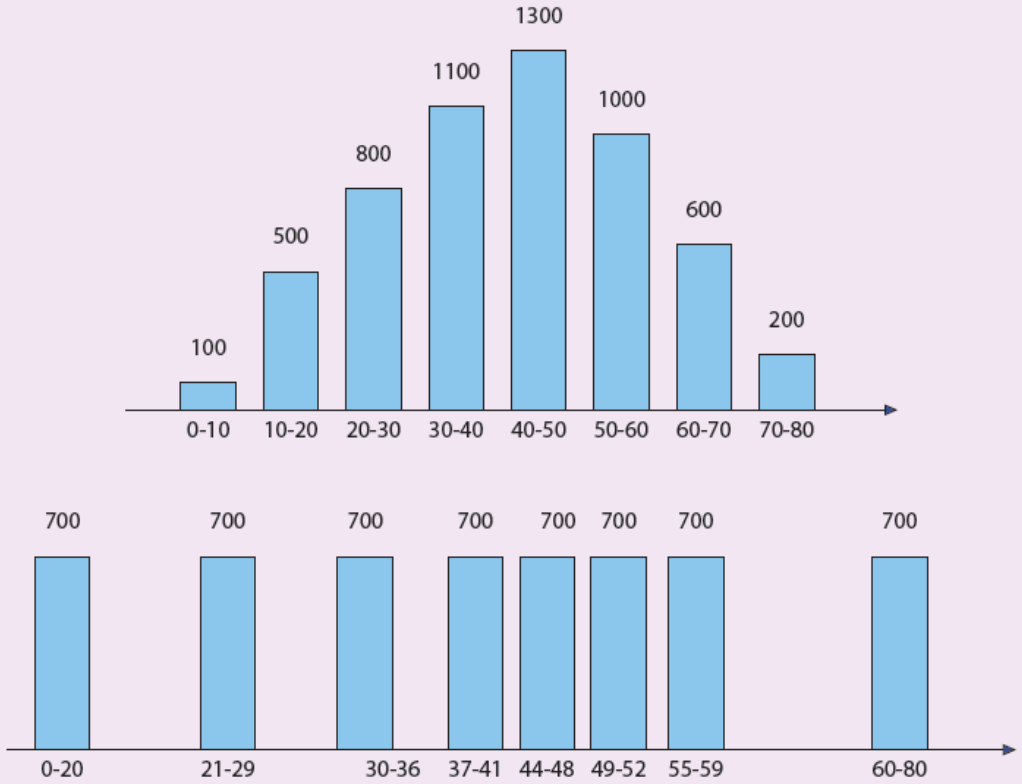
Example (Equi-depth histogram ($B = 4, d = 16$))



Comparison

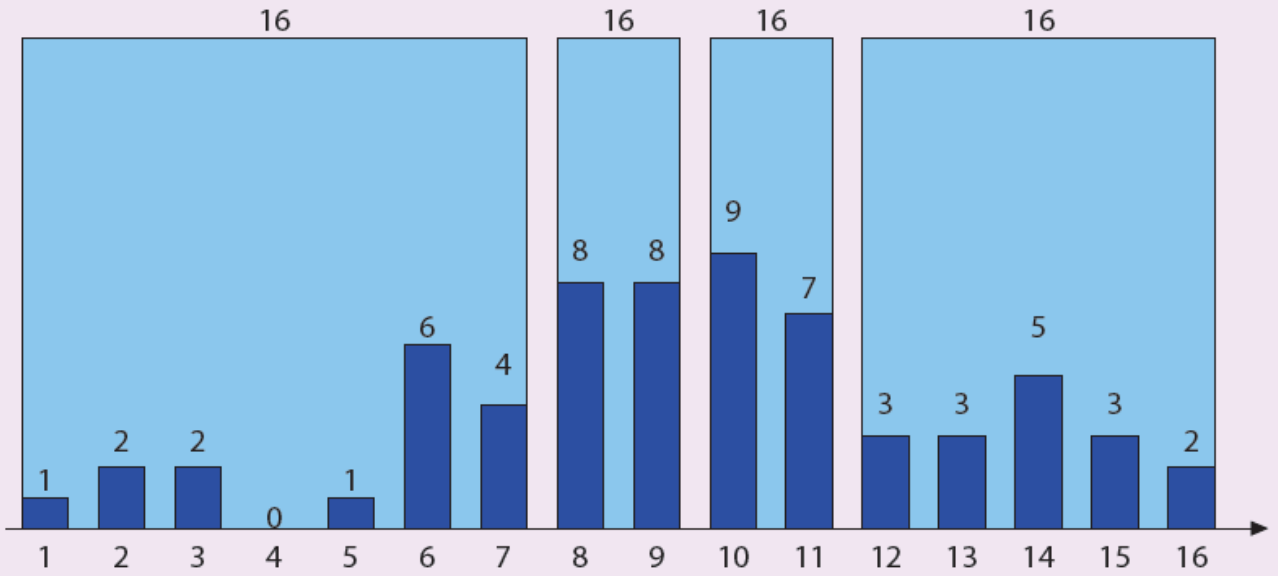
Equi-depth histogram “invests” bytes in the densely populated customer age region between 30 and 59

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



Equality Selection

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

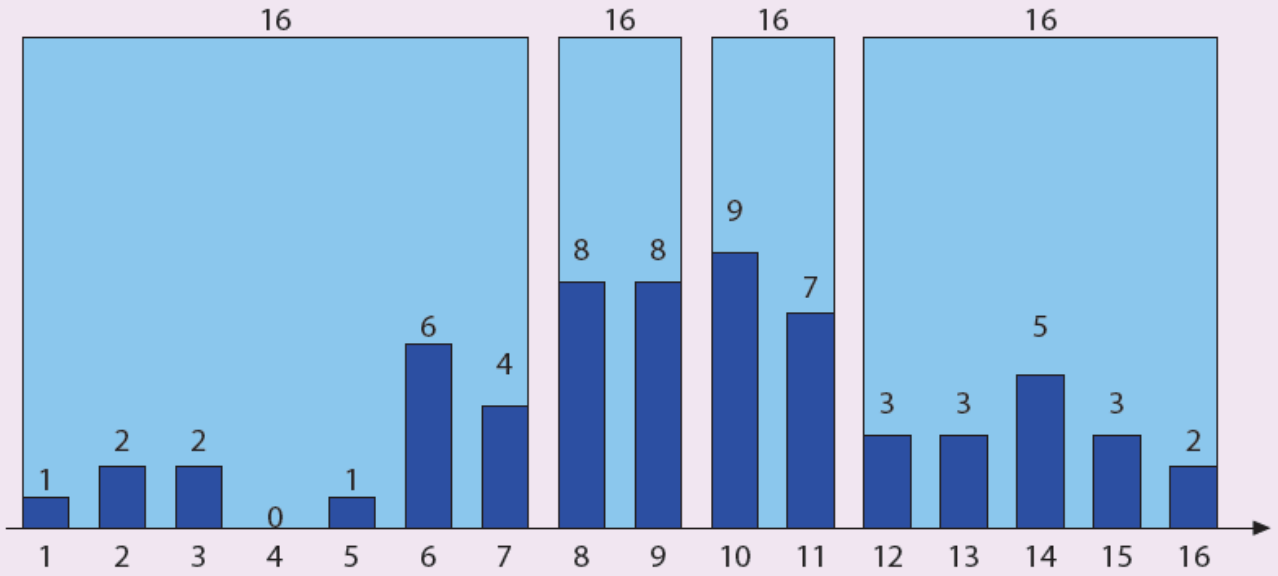


- Value 5 is in first bucket $[1, 7]$ (with $d = 16$ tuples)
- Assume **uniform distribution within the bucket**:

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

Range Selection

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



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

$$|Q| = 16 + 16 + 16 + {}^{2}/_{7} \cdot 16 \approx 53 \quad .$$

Conclusion

Query rewriting

- The DBMS can identify better query plans even without a cost model
- Filtering as early as possible is usually a good choice

Selectivity estimation

- Uniformity & independence assumptions
- Histograms
- Join selectivity

Next: Now that we can (roughly) estimate the selectivity of predicates, what can we actually do with them?