CMSC424: Database Design

Instructor: Amol Deshpande
amol@cs.umd.edu

Spring 2020 – Online Instruction Plan

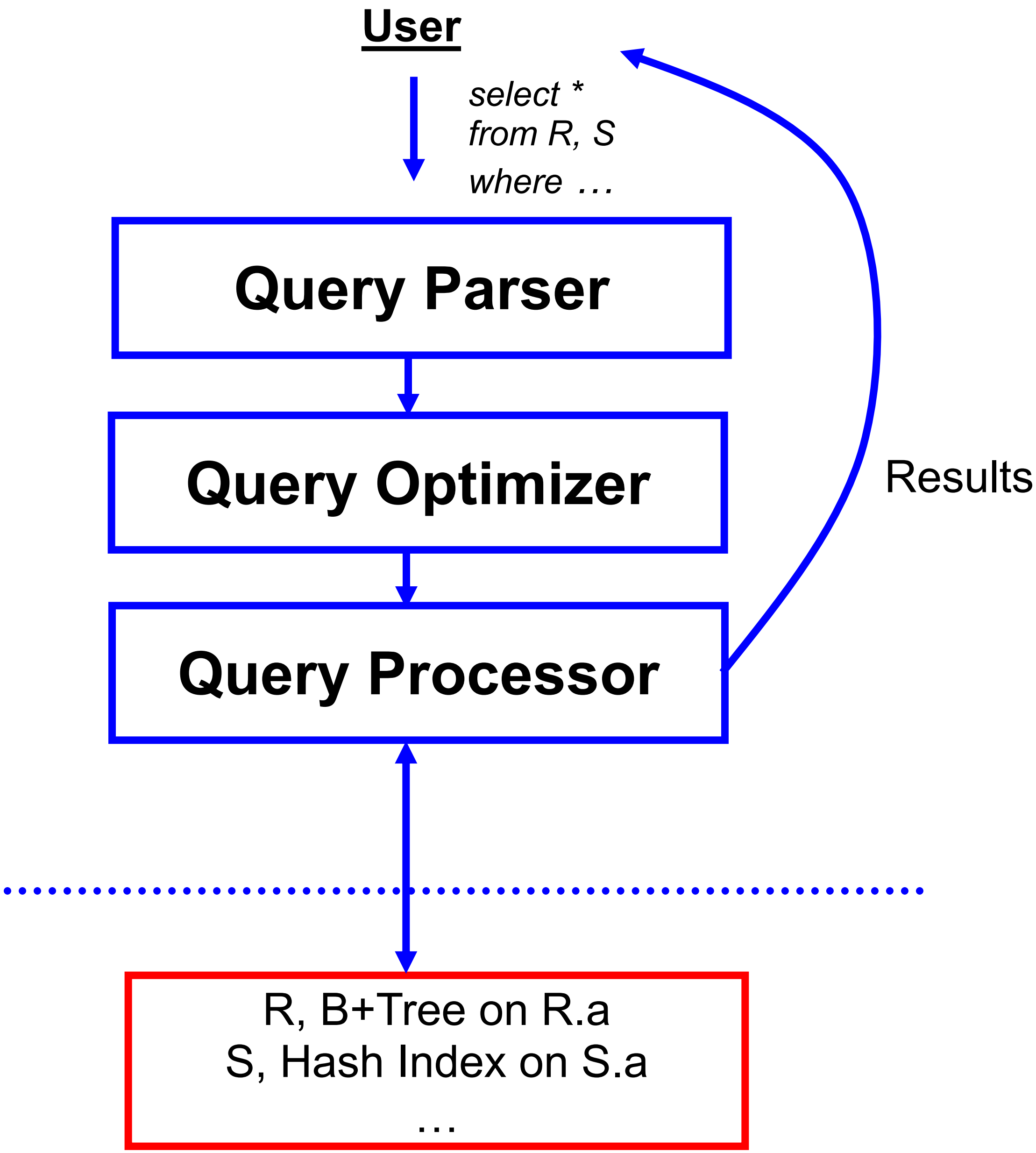
Modified to swap the last two projects

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3: Query Optimization; Architectures/Parallel 1
- Week 4: Parallel Databases + MapReduce; Transactions 1
- Week 5: Transactions 2

Spring 2020 – Online Instruction Plan

- Week 1: File Organization and Indexes
- Week 2: Query Processing
- Week 3 (Homework Due April 17, Noon)
  - Query Optimization 1: Overview, Statistics
  - Query Optimization 2: Equivalences, Search Algorithms
  - Architectures/Parallel Databases Introduction
- Week 4: Parallel Databases; Mapreduce; Transactions 1
  - Map-reduce and Apache Spark (will post early for Project 5)
- Week 5: Transactions 2

Getting Deeper into Query Processing

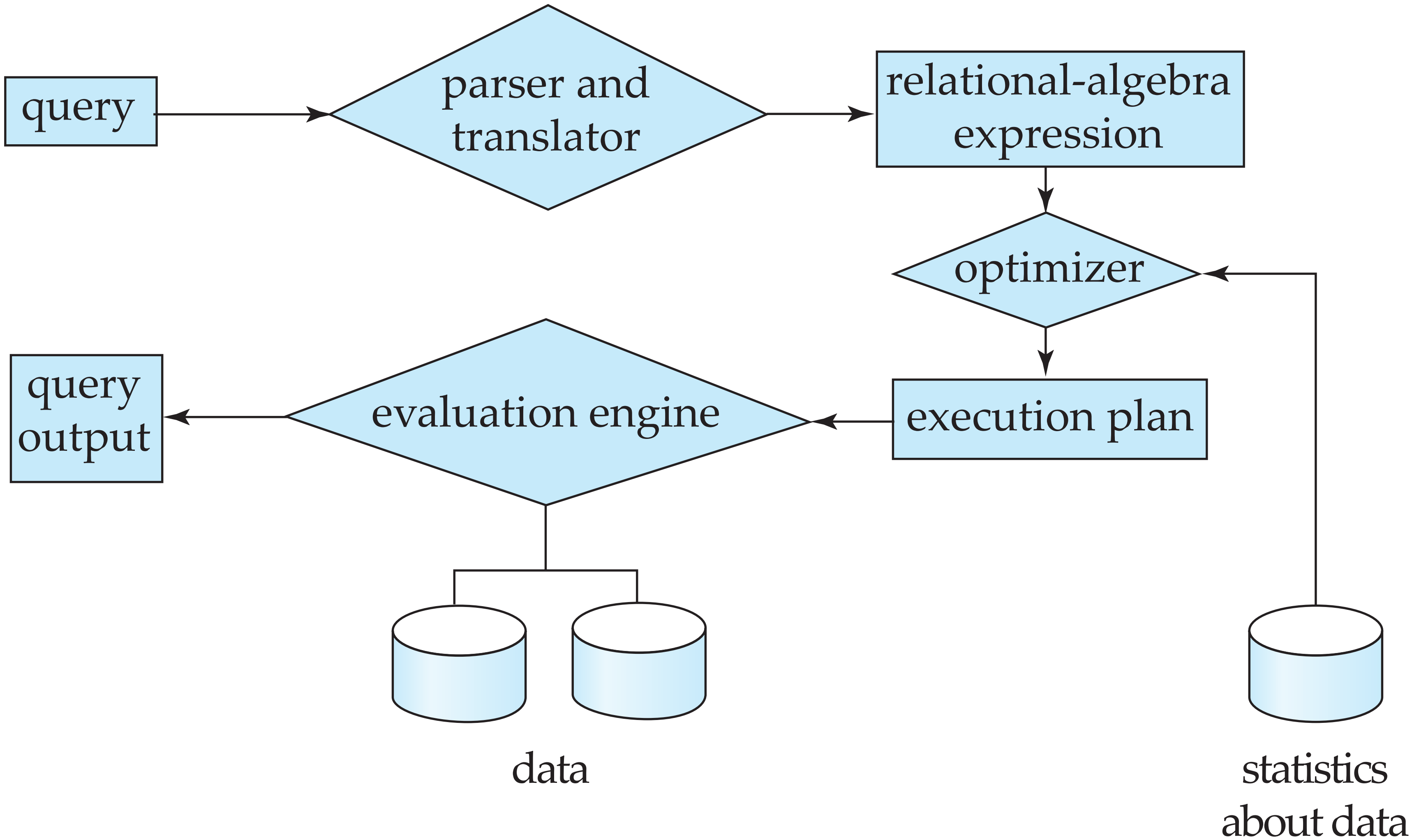


Resolve the references,
Syntax errors etc.
Converts the query to an internal format
relational algebra like

Find the *best* way to evaluate the query
- Which index to use ?
- What join method to use ?
- …

Read the data from the files
Do the query processing
- *joins, selections, aggregates*
- …

Getting Deeper into Query Processing



Query Optimization

- Book Chapters
  - 13.1, 13.3
- Key topics:
  - Why query optimization is so important?
  - How to estimate the sizes of “intermediate results”
  - Histograms for estimating sizes of selections
  - Brief discussion of intermediate sizes of other operators

Query Optimization

- Overview
- Statistics Estimation
- Transformation of Relational Expressions
- Optimization Algorithms

Query Optimization

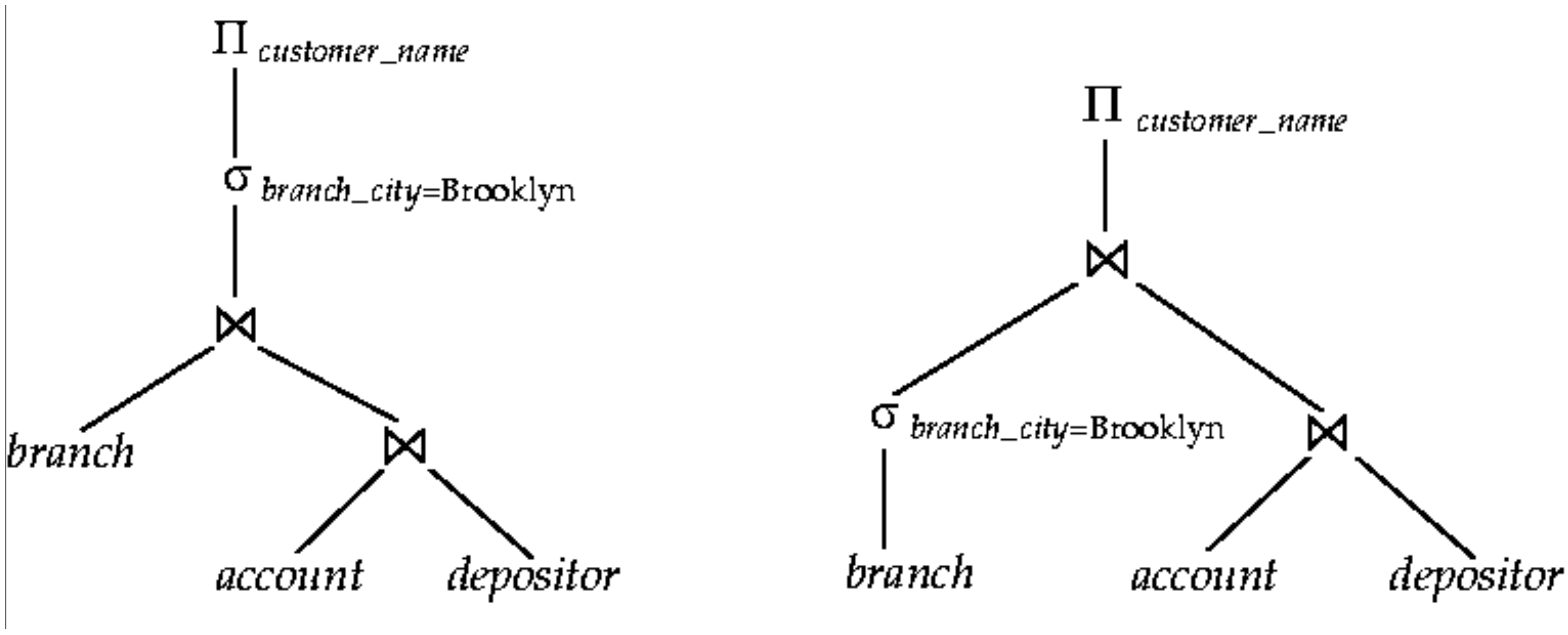
- Why ?
  - Many different ways of executing a given query
  - Huge differences in cost
- Example:
  - select * from person where ssn = “123”
  - Size of *person* = 1GB
  - Sequential Scan:
    - Takes 1GB / (20MB/s) = 50s
  - Use an index on SSN (assuming one exists):
    - Approx 4 Random I/Os = 40ms

Query Optimization

- Many choices
  - Using indexes or not, which join method (hash, vs merge, vs NL)
  - What join order ?
    - Given a join query on R, S, T, should I join R with S first, or S with T first ?
- This is an optimization problem
  - Similar to say *traveling salesman problem*
  - Number of different choices is very very large
  - Step 1: Figuring out the *solution space*
  - Step 2: Finding algorithms/heuristics to search through the solution space

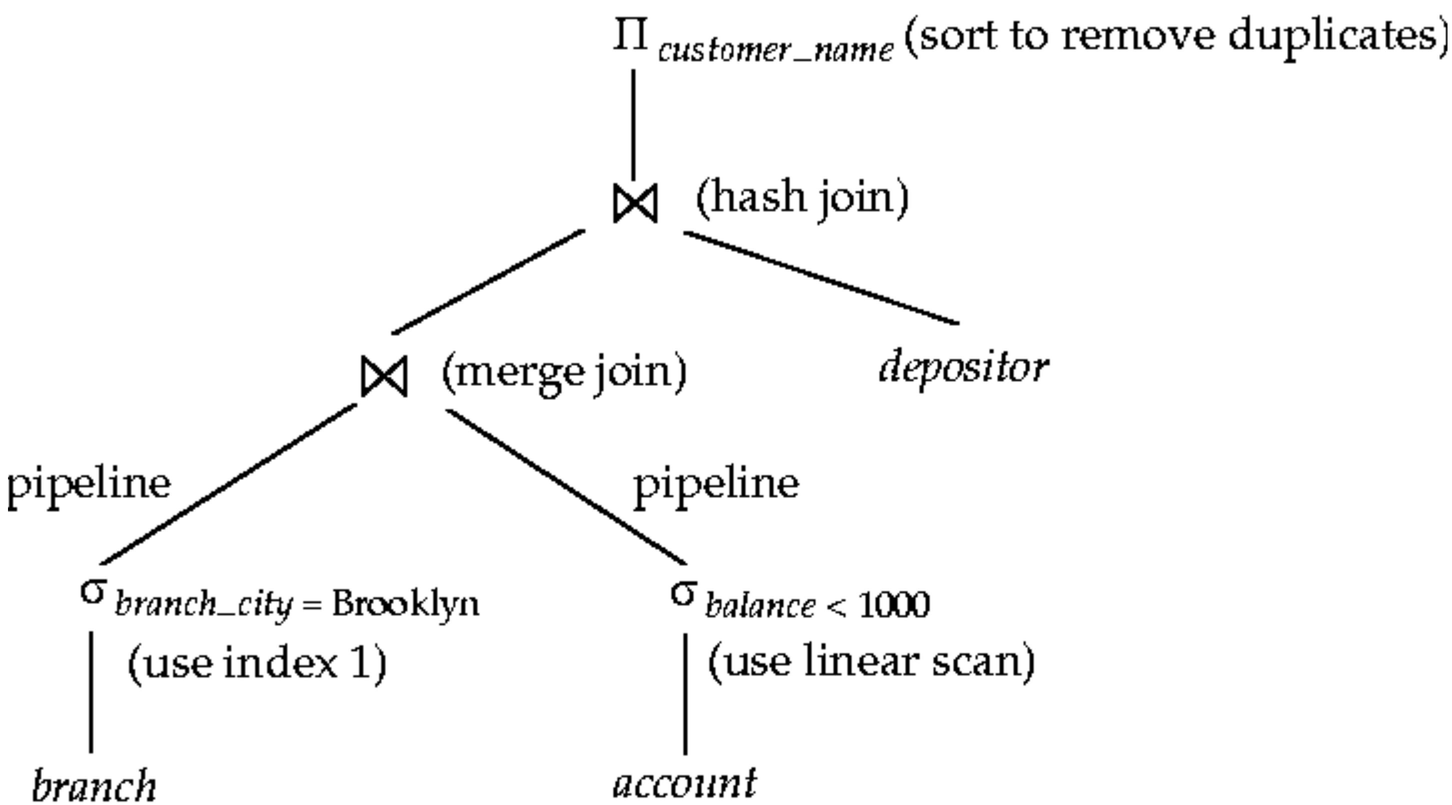
Query Optimization

- Equivalent relational expressions
  - Drawn as a tree
  - List the operations and the order



Query Optimization

- Execution plans
  - Evaluation expressions annotated with the methods used



Query Optimization

- Steps:
  - Generate all possible execution plans for the query
  - Figure out the cost for each of them
  - Choose the best
- Not done exactly as listed above
  - Too many different execution plans for that
  - Typically interleave all of these into a single efficient search algorithm

Query Optimization

- Steps:
  - Generate all possible execution plans for the query
    - First generate all equivalent expressions
    - Then consider all annotations for the operations
  - Figure out the cost for each of them
    - Compute cost for each operation
      - Using the formulas discussed before
      - One problem: How do we know the number of result tuples for, say, $\sigma_{balance<2500}(account)$
    - Add them !
  - Choose the best

Query Optimization

- Introduction
- Statistics Estimation
- Transformation of Relational Expressions
- Optimization Algorithms

Cost estimation

- Computing operator costs requires information like:
  - Primary key ?
  - Sorted or not, which attribute
    - So we can decide whether need to sort again
  - How many tuples in the relation, how many blocks ?
  - RAID ?? Which one ?
    - Read/write costs are quite different
  - How many tuples match a predicate like “age > 40” ?
    - E.g. Need to know how many index pages need to be read
  - Intermediate result sizes
    - E.g. (R JOIN S) is input to another join operation – need to know if it fits in memory
  - And so on…

Cost estimation

- Some information is static and is maintained in the metadata
  - Primary key ?
  - Sorted or not, which attribute
    - So we can decide whether need to sort again
  - How many tuples in the relation, how many blocks ?
  - RAID ?? Which one ?
    - Read/write costs are quite different
- Typically kept in some tables in the database
  - “all_tab_columns” in Oracle
- Most systems have commands for updating them

Cost estimation

- However, others need to be estimated somehow
  - How many tuples match a predicate like “age > 40” ?
    - E.g. Need to know how many index pages need to be read
  - Intermediate result sizes
- The problem variously called:
  - “intermediate result size estimation”
  - “selectivity estimation”

- Very important to estimate reasonably well
  - e.g. consider “select * from R where zipcode = 20742”
  - We estimate that there are 10 matches, and choose to use a secondary index (remember: random I/Os)
  - Turns out there are 10000 matches
  - Using a secondary index very bad idea
  - Optimizer also often choose Nested-loop joins if one relation very small… underestimation can result in very bad

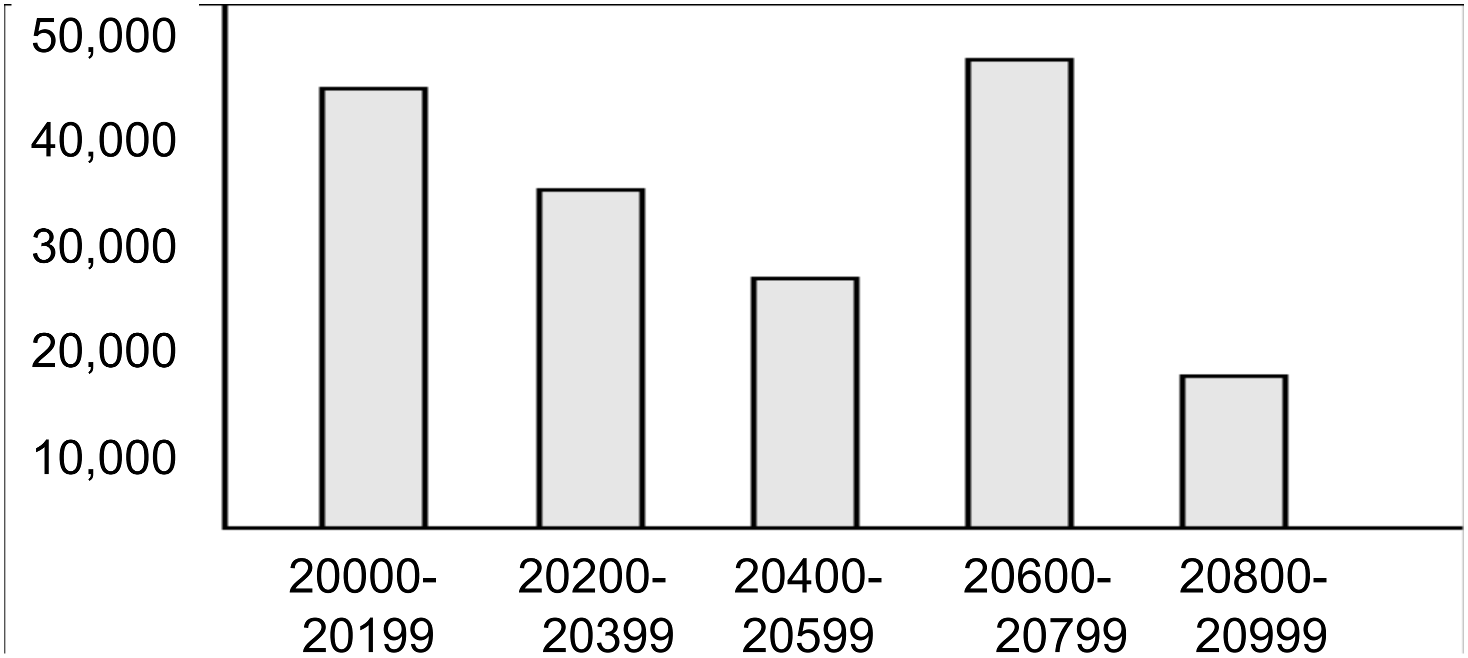
Selectivity Estimation

- Basic idea:
  - Maintain some information about the tables
    - More information → more accurate estimation
    - More information → higher storage cost, higher update cost
  - Make uniformity and randomness assumptions to fill in the gaps

- Example:
  - For a relation "people", we keep:
    - Total number of tuples = 100,000
    - Distinct "zipcode" values that appear in it = 100
  - Given a query: "zipcode = 20742"
    - We estimated the number of matching tuples as: 100,000/100 = 1000
  - What if I wanted more accurate information ?
    - Keep histograms…

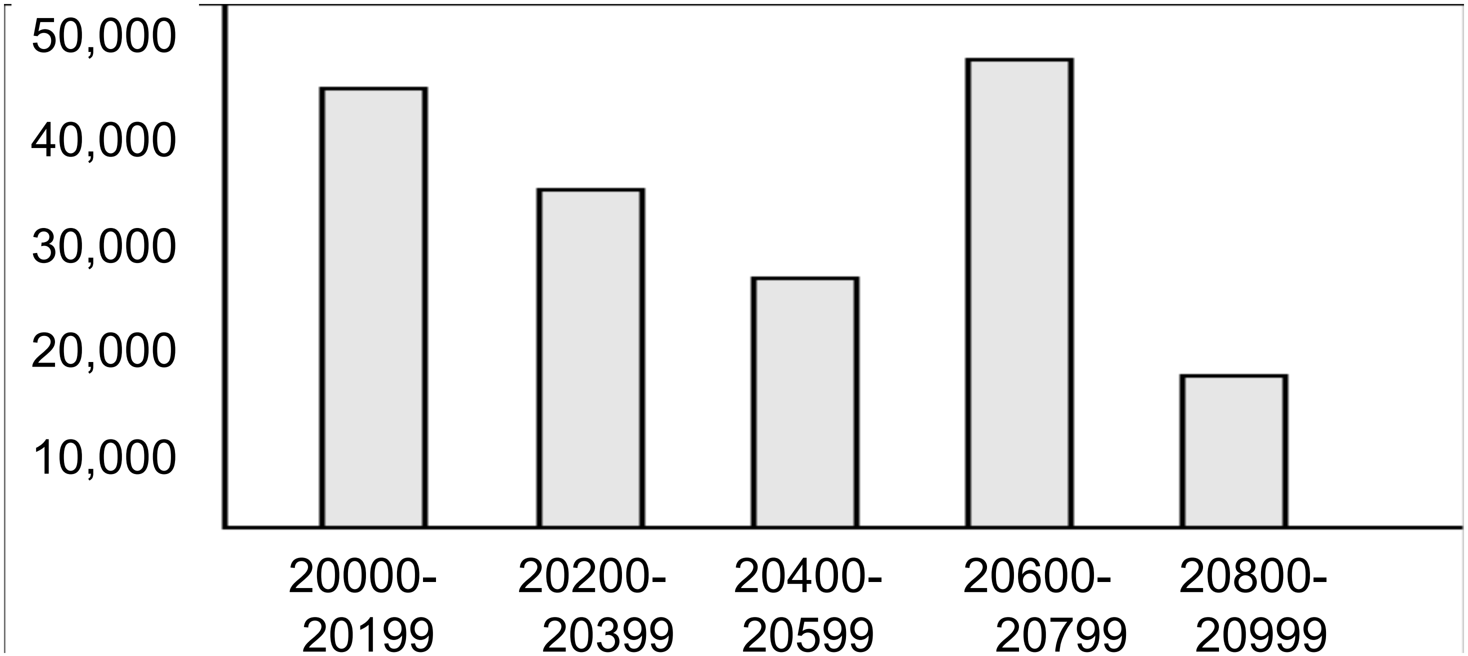
Histograms

- A condensed, approximate version of the "frequency distribution"
  - Divide the range of the attribute value in "buckets"
  - For each bucket, keep the total count
  - Assume uniformity within a bucket



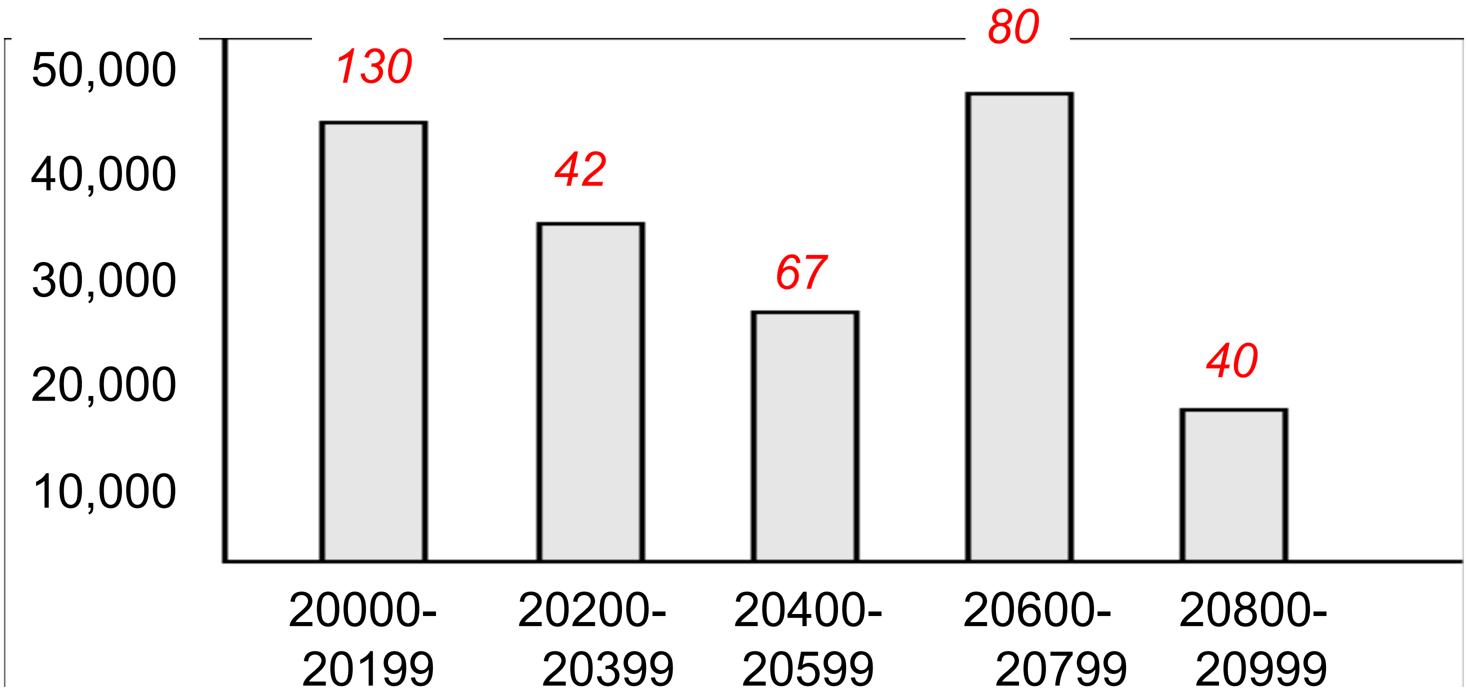
Histograms

- Given a query: zipcode = “ 20742”
  - Find the bucket (Number 3)
  - Say the associated cound = 45000
  - Assume uniform distribution within the bucket: 45,000/200 = 225



Histograms

- What if the ranges are typically not full ?
  - ie., only a few of the zipcodes are actually in use ?
- With each bucket, also keep the number of zipcodes that are valid
- Now the estimate would be: 45,000/80 = 562.50
- More Information → Better estimation



Histograms

- Very widely used in practice
  - One-dimensional histograms kept on almost all columns *of interest*
    - ie., the columns that are commonly referenced in queries
  - Sometimes: multi-dimensional histograms also make sense
    - Less commonly used as of now
- Two common types of histograms:
  - Equi-depth
    - The attribute value range partitioned such that each bucket contains about the same number of tuples
  - Equi-width
    - The attribute value range partitioned in equal-sized buckets
  - VOptimal histograms
    - No such restrictions
    - More accurate, but harder to use or update

Next…

- Estimating sizes of the results of various operations
- Guiding principle:
  - Use all the information available
  - Make uniformity and randomness assumptions otherwise
  - Many formulas, but not very complicated…
    - In most cases, the first thing you think of

Basic statistics

- Basic information stored for all relations
  - n_r: number of tuples in a relation *r.*
  - b_r: number of blocks containing tuples of *r.*
  - l_r: size of a tuple of *r.*
  - f_r: blocking factor of *r* — i.e., the number of tuples of *r* that fit into one block.
  - *V(A, r):* number of distinct values that appear in *r* for attribute *A;* same as the size of $\prod_A(r)$.
  - *MAX(A, r):* th maximum value of *A* that appears in *r*
  - *MIN(A, r)*
  - If tuples of *r* are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Selection Size Estimation

- $\sigma_{A=v}(r)$
  - $n_r / V(A,r)$: number of records that will satisfy the selection
  - Equality condition on a key attribute: *size estimate* = 1

- $\sigma_{A \leq V}(r)$ (case of $\sigma_{A \geq V}(r)$ is symmetric)
  - Let c denote the estimated number of tuples satisfying the condition.
  - If min(A,r) and max(A,r) are available in catalog
    - c = 0 if v < min(A,r)
    - c = $n_r \cdot \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
  - If histograms available, can refine above estimate
  - In absence of statistical information *c* is assumed to be $n_r / 2$.

Size Estimation of Complex Selections

- **selectivity(θ_i)** = the probability that a tuple in r satisfies θ_i .
  - If s_i is the number of satisfying tuples in r, then selectivity (θ_i) = s_i / n_r.

- **Conjunction:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$. *Assuming independence,* estimate of tuples in the result is:

$$n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$$

- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$. Estimated number of tuples:

$$n_r * \left(1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r}) \right)$$

- **Negation:** $\sigma_{\neg\theta}(r)$. Estimated number of tuples: $n_r - size(\sigma_\theta(r))$

Joins

- R JOIN S: R.a = S.a
  - |R| = 10,000; |S| = 5000
- CASE 1: *a* is key for S
  - *Each tuple of R joins with exactly one tuple of S*
  - So: |R JOIN S| = |R| = 10,000
  - Assumption: Referential integrity holds
  - What if there is a selection on R or S
    - Adjust accordingly
    - Say: S.b = 100, with selectivity 0.1
    - THEN: |R JOIN S| = |R| * 0.1 = 100
- CASE 2: *a* is key for R
  - Similar

Joins

- R JOIN S: R.a = S.a
  - |R| = 10,000; |S| = 5000

- CASE 3: *a* is not a key for either
  - Reason with the distributions on *a*
  - Say: the domain of *a: V(A, R)* = 1000 (the number of distinct values *a* can take)
  - THEN, *assuming uniformity*
    - For each value of *a*
      - We have 10,000/100 = 100 tuples of R with that value of a
      - We have 5000/100 = 50 tuples of S with that value of a
      - All of these will join with each other, and produce 100 *50 = 5000
    - So total number of results in the join:
      - 5000 * 100 = 500000
  - We can improve the accuracy if we know the distributions on *a* better
    - Say using a histogram

Other Operations

- Projection: $\prod_A(R)$
  - If no duplicate elimination, THEN $|\prod_A(R)| = |R|$
  - If *distinct* used (duplicate elimination performed): $|\prod_A(R)| = V(A, R)$

- Set operations:
  - Union ALL: $|R \cup S| = |R| + |S|$
  - Intersect ALL: $|R \cap S| = \min\{|R|, |S|\}$
  - Except ALL: $|R - S| = |R|$ (a good upper bound)
  - Union, Intersection, Except (with duplicate elimination)
    - Somewhat more complex reasoning based on the frequency distributions etc…

- And so on …