Relational Database System Implementation

CS122 – Lecture 9

Winter Term, 2018-2019

Last Time: Plan Costing

- Introduced the notion of plan costing
- Goal: Faster plans end up with lower cost than slower ones
- Need to collect statistics on tables in order to make cost estimates
- A basic, minimal set of statistics:
  - n_r – the number of tuples in table r
  - b_r – the number of blocks containing tuples in r
  - l_r – the average size of a tuple in r, in bytes
  - V(A, r) – number of distinct values of A in table r
  - min(A, r) – minimum value of A in table r
  - max(A, r) – maximum value of A in table r

Select Costs

- $\sigma_\theta(r)$
- Estimate number of rows produced $n_\sigma = n_r \times P(\theta)$
  - $P(\theta)$ is the *selectivity* of the predicate
  - i.e. the likelihood that a tuple will satisfy the predicate
- Simply need to estimate the selectivity of the predicate, then we can estimate the number of rows produced
- For now, assume that *r* is a heap file
  - Select operation will [almost] always read all blocks in *r*
  - *(Other file organizations and indexes change this…)*

Selectivity of Simple Predicates

- $\sigma_{A \le v}(r)$
  - Without a histogram, use minimum/maximum values for A to estimate selectivity
- If $v < \min(A, r)$:
  - $P(A \le v) = 0$
- If $v > \max(A, r)$:
  - $P(A \le v) = 1$
- If $\min(A, r) \le v \le \max(A, r)$:
  - $P(A \le v) = (v - \min(A, r)) / (\max(A, r) - \min(A, r))$
- $\sigma_{A \ge v}(r)$ is similar

Selectivity of Simple Predicates (2)

- $\sigma_{A=v}(r)$
  - Assume uniform distribution of different values of A
  - Estimate $P(A=v)$ to be $1 / V(A, r)$
  - Estimate $n_\sigma = n_r / V(A, r)$
- What if A is a primary key for r ?
  - In that case, $V(A, r)$ will be n_r
  - $P(A=v)$ will be $1 / n_r$, and n_σ will be 1

Selectivity of Simple Predicates (3)

- $\sigma_{A=v}(r)$

- If A is a primary key for r, can also improve file-scan performance:
  - Each value of A can only appear once…
  - Stop scanning r when we find the specified row
  - Average-case block-reads = b_r / 2; worst-case = b_r

Selectivity of Simple Predicates (4)

- For inverse of these predicates: $\sigma_{A>v}(r)$, $\sigma_{A\neq v}(r)$
  - Simply compute selectivity as 1 – P($A \leq v$) or 1 – P($A = v$)
- Boolean negation can be handled in similar way:
  - $\sigma_{\neg\theta}(r)$
  - Simple: $P(\neg\theta) = 1 - P(\theta)$

Complex Selects

- If a predicate includes multiple conditions, estimate selectivities of the components, then combine
- Conjunctive selections: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots}(r)$
  - Assumption: conditions are independent of each other
  - $P(\theta 1 \wedge \theta 2 \wedge \ldots) = P(\theta 1) \times P(\theta 2) \times P(\ldots)$
- Disjunctive selections: $\sigma_{\theta 1 \vee \theta 2 \vee \ldots}(r)$
  - Again, compute selectivities of components
  - $P(\theta 1 \vee \theta 2 \vee \ldots)$ = probability that a tuple satisfies at least one condition = 1 – probability it satisfies *none* of them
  - $P(\theta 1 \vee \theta 2 \vee \ldots) = 1 - (1 - P(\theta 1)) \times (1 - P(\theta 2)) \times \ldots$

Estimating Selectivity

- One major assumption here:
  - Conditions involve simple comparisons between an attribute and a constant
- Frequently not true!
  - SELECT * FROM employees WHERE salary * 1.05 > 100000;
  - DELETE FROM employees
    WHERE compute_popularity(emp_id) < 20;
- In simpler cases, can analyze expression to make estimate
- For more difficult situations, use default selectivities, e.g.
  - 1/2 when it's expected to be "common" for tuples to satisfy the condition
  - 1/3 or 1/4 when it's expected to be "uncommon" or "rare"

Selection Against Subplans

- Previous examples were all against a relation *r*
  - *We had statistics for r!*
- Plans often contain selections against subplans
- Need to estimate the statistics of a plan-node's result as well, if higher-level cost estimates will be useful
- Most difficult are $V(A, r)$, $\min(A, r)$, and $\max(A, r)$
- If selection involves an equality: $\sigma_{A=v}(r)$
  - $V(A, \sigma_{A=v}) = 1$
  - $\min(A, \sigma_{A=v}) = \max(A, \sigma_{A=v}) = v$

Selection Against Subplans (2)

- If selection involves a comparison: $\sigma_{A \le v}(r)$
  - Assume $\min(A, r) \le v \le \max(A, r)$
  - $\min(A, \sigma_{A \le v}) = \min(A, r)$
  - $\max(A, \sigma_{A \le v}) = v$
  - Estimate $V(A, \sigma_{A \le v})$
    $= V(A, r) \times (v - \min(A, r)) / (\max(A, r) - \min(A, r))$
    $= V(A, r) \times P(A \le v)$

- In general, if θ is *A op v*:
  - *op* is some inequality comparison: $< \; > \; \le \; \ge \; \ne$
  - Estimate $V(A, \sigma_\theta) = V(A, r) \times P(\theta)$

Selection Against Subplans (3)

- If predicate θ forces *A* to take on a set of values:
  - SELECT * FROM schedule WHERE hour = 3 OR hour = 4;
  - SELECT * FROM shapes
    WHERE color IN ('red', 'orange', 'yellow');
  - $V(A, \sigma_\theta)$ = number of values in the predicate
  - Can compute $\min(A, \sigma_\theta)$, $\max(A, \sigma_\theta)$ from these as well
- If none of these situations occur:
  - Assume $V(A, \sigma_\theta)$, $\min(A, \sigma_\theta)$, $\max(A, \sigma_\theta)$ are independent of selection criteria!
  - Set $V(A, \sigma_\theta)$ to $\min(V(A, r), n_\sigma)$
    - # of distinct values for *A* is capped by # of rows produced by σ

Join Costs

- Several important costs to estimate for joins
  - Number of rows produced by the join operation
  - Number of disk IOs performed by the join operation
- Second value is harder to estimate, primarily due to the buffer manager, but still critical to estimate
- Example: nested loop join (no optimizations)
  - Worst case (unlikely): $b_r + n_r \times b_s$ block reads
  - Best case (inner table fits in memory): $b_r + b_s$ reads
- Disk IO estimate is very approximate, and depends on the specific join implementation being used

Join Costs (2)

- For now, focus on the number of rows produced
- Cartesian product: $r \times s$
  - Every row in table *r* is joined to every row in table *s*
  - $n_{r \times s} = n_r \times n_s$
  - Average tuple length $l_{r \times s} = l_r + l_s$
- Theta join: $r \bowtie_\theta s$
  - Can model as $\sigma_\theta(r \times s)$; compute estimates as for $\sigma_\theta(\ldots)$
  - Big problem: our cost estimates are most accurate when comparing attributes to constants!
  - Join predicates usually compare attributes to attributes

Join Costs (3)

- To compute proper join estimates, need to look at the attributes being compared
- For theta-join $r \bowtie_{r.A=s.A} s$:
  - If $r.A$ is a key for r:
    - Each tuple in s will join with at most one tuple in r
    - Estimate number of tuples in result $n_{r \bowtie s} = n_s$
  - Similarly, if $s.A$ is a key for s:
    - Each tuple in r will join with at most one tuple in s
    - Estimate $n_{r \bowtie s} = n_r$
  - If both are keys for their respective tables:
    - $n_{r \bowtie s} = \min(n_r, n_s)$

Join Costs (4)

- For theta-join $r \bowtie_{r.A=s.A} s$:
  - If neither *r.A* nor *s.A* is a key for its respective table:
    - Assume that *A* is uniformly distributed in both *r* and *s*
    - *(Note: ignoring min/max stats for these estimates)*
  - Given a specific tuple t_r in *r*, estimate that n_s / V(*A*, *s*) tuples in *s* will join with that tuple
    - n_s × probability that a given tuple t_s in *s* will have value $t_r.A$
    - Suggests that $n_{r \bowtie s} = n_r \times n_s / V(A, s)$
  - But, given a specific tuple t_s in *s*, estimate n_r / V(*A*, *r*) tuples in *r* will join with that tuple
    - Suggests that $n_{r \bowtie s} = n_s \times n_r / V(A, r)$

Join Costs (5)

- For theta-join $r \bowtie_{r.A=s.A} s$:
  - Two estimates for number of rows produced:
    - $n_{r \bowtie s} = n_r \times n_s \,/\, \mathrm{V}(A, s)$ *(from perspective of tuples in r)*
    - $n_{r \bowtie s} = n_s \times n_r \,/\, \mathrm{V}(A, r)$ *(from perspective of tuples in s)*
  - If $\mathrm{V}(A, r) < \mathrm{V}(A, s)$:
    - Expect that more tuples in *s* will not join with any tuple in *r*
    - Use estimate based on *r*: $n_{r \bowtie s} = n_r \times n_s \,/\, \mathrm{V}(A, s)$
    - Similarly, if $\mathrm{V}(A, r) > \mathrm{V}(A, s)$, more tuples in *r* will be left out
  - If $\mathrm{V}(A, r) \neq \mathrm{V}(A, s)$, choose the larger of $\mathrm{V}(A, r)$, $\mathrm{V}(A, s)$
  - Estimate $n_{r \bowtie s} = n_r \times n_s \,/\, \max(\mathrm{V}(A, r), \mathrm{V}(A, s))$

Join Costs (6)

- Can extend these estimates to joins with multiple conjuncts
- For theta-join $r \bowtie_{r.A=s.A \wedge r.B=s.B} s$:
  - Check if ($r.A$, $r.B$) or any proper subset is a key for r
  - Check if ($s.A$, $s.B$) or any proper subset is a key for s
  - If so, compute estimates as before
- If attributes are *not* keys for r or s:
  - Again, assume the conditions are independent of each other
  - $\text{P}(r.A=s.A \wedge r.B=s.B) = \text{P}(r.A=s.A) \times \text{P}(r.B=s.B)$
    $= 1 / (\max(\text{V}(A, r), \text{V}(A, s)) \times \max(\text{V}(B, r), \text{V}(B, s)))$
  - $n_{r \bowtie s} = n_r \times n_s / (\max(\text{V}(A, r), \text{V}(A, s)) \times \max(\text{V}(B, r), \text{V}(B, s)))$

Outer Join Costs

- Can use very simple estimates for outer joins
  - Again, only using number of distinct values; not using min/max to further refine statistics
- Left outer join: $n_{r \text{⟕} s} = n_{r \bowtie s} + n_r$
- Right outer join: $n_{r \text{⟖} s} = n_{r \bowtie s} + n_s$
- Full outer join: $n_{r \text{⟗} s} = n_{r \bowtie s} + n_r + n_s$
- These estimates are almost certainly much higher than actual row-counts will be, but they are an upper bound
  - …and they are fast to compute.
  - Could devise a better estimate, but really want to move to better stats (e.g. storing histograms) to make it worthwhile

Other Plan Nodes

- Project: $\Pi_{\ldots}(r)$
- $\Pi_A(r)$, where A is a simple column-reference
  - $n_\Pi = n_r$ (no duplicate-elimination in SQL)
  - $V(A, \Pi_A) = V(A, r)$
  - Similarly, min/max don't change
- $\Pi_E(r)$, where E is an expression possibly with functions
  - Again, $n_\Pi = n_r$
  - For $V(E, \Pi_E)/\min(E, \Pi_E)/\max(E, \Pi_E)$, no idea! Either need to guess, or we need more knowledge about E.
    - E.g. just guess $V(E, \Pi_E) = n_\Pi$

Other Plan Nodes (2)

- Grouping/aggregation: ${}_{G1,G2,\ldots}\mathcal{G}_{E1,E2,\ldots}(r)$
  - Gi can be either column-references or expressions
  - Ei can be simple aggregate function calls, or more advanced expressions involving aggregate functions
    - SELECT SUM(CASE WHEN a < b THEN 1 ELSE 0 END) FROM t;
    - SELECT MIN(a) + MAX(b) FROM t;
- For simple column-references in grouping attributes:
  - $n_{\mathcal{G}} = V(G1, r) \times V(G2, r) \times \ldots$
  - $V(G1, \mathcal{G}) = V(G1, r)$, etc.

Other Plan Nodes (3)

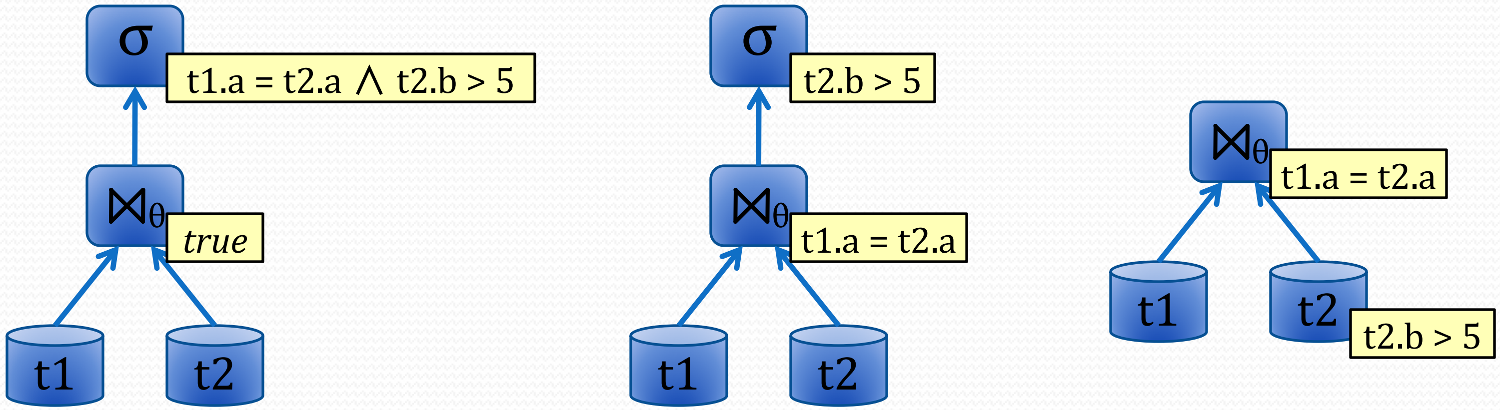
- Grouping/aggregation: $_{G1,G2,...}\mathcal{G}_{E1,E2,...}(r)$
- For simple column-references and simple aggregates:
  - Guess COUNT(A), SUM(A), AVG(A) will produce different values for each group. e.g. V(COUNT(A), $\mathcal{G}$) = $n_{\mathcal{G}}$
- Can be a bit more clever with MIN(A) and MAX(A)
  - Could guess V(MIN(A), $\mathcal{G}$) = $n_{\mathcal{G}}$ as before
  - Note that MIN(A)/MAX(A) will always select an *existing* value of A from input relation
  - A better guess: V(MIN(A), $\mathcal{G}$) = min(V(A, r), $n_{\mathcal{G}}$)

Summary – Plan Costing

- Plan costing is a very imprecise process
  - Almost certainly inaccurate, except in *very* simple cases
  - Hopefully estimates are "good enough" to guide plan selection
  - *(Most databases provide ways to give the optimizer hints about plan optimization)*
- These estimates are simply one way of estimating costs
  - Different assumptions, or different kinds of statistics, will produce different costing estimates
- Still, an essential part of query planning!
  - Collecting useful table stats, then making reasonably accurate estimates from them, greatly improves DB query performance
  - *(Becomes very obvious when table stats are inaccurate)*

Equivalent Plans?

- Previously had this query:
  - SELECT * FROM t1, t2 WHERE t1.a = t2.a AND t2.b > 5;



- How do we know these plans are actually equivalent?

Equivalent Plans

- Two plans are *equivalent* if they produce the same results for every legal database instance
  - A “legal” database instance satisfies all constraints
- Generally, the order of tuples is irrelevant
  - If sorting is not specified on results, two equivalent plans may generate results in different orders
- *Equivalence rules* specify different forms of an expression that are equivalent
  - Can prove that these rules hold for all legal databases
  - Can use them to transform query plans into equivalent (but hopefully faster) plans

Simple Equivalence Rules

- Cascade of σ:
  - $\sigma_{\theta1\wedge\theta2}(E) = \sigma_{\theta1}(\sigma_{\theta2}(E))$
- σ is commutative:
  - $\sigma_{\theta1}(\sigma_{\theta2}(E)) = \sigma_{\theta2}(\sigma_{\theta1}(E))$
- Selections, Cartesian products, and theta-joins:
  - $\sigma_{\theta}(E1 \times E2) = E1 \bowtie_{\theta} E2$
  - $\sigma_{\theta1}(E1 \bowtie_{\theta2} E2) = E1 \bowtie_{\theta1\wedge\theta2} E2$
- Theta-joins are commutative:
  - $E1 \bowtie_{\theta} E2 = E2 \bowtie_{\theta} E1$

Theta Join Equivalence Rules

- Natural joins are associative:
  - $(E1 \bowtie E2) \bowtie E3 = E1 \bowtie (E2 \bowtie E3)$
- Theta-joins are also associative, but it's a bit trickier:
  - $(E1 \bowtie_{\theta 1} E2) \bowtie_{\theta 2 \wedge \theta 3} E3 = E1 \bowtie_{\theta 1 \wedge \theta 3} (E2 \bowtie_{\theta 2} E3)$
  - θ1 only refers to attributes in E1 and/or E2
  - θ2 only refers to attributes in E2 and/or E3
  - θ3 only refers to attributes in E1 and/or E3
  - Any of these conditions might also simply be *true*

Theta Join Equivalence Rules (2)

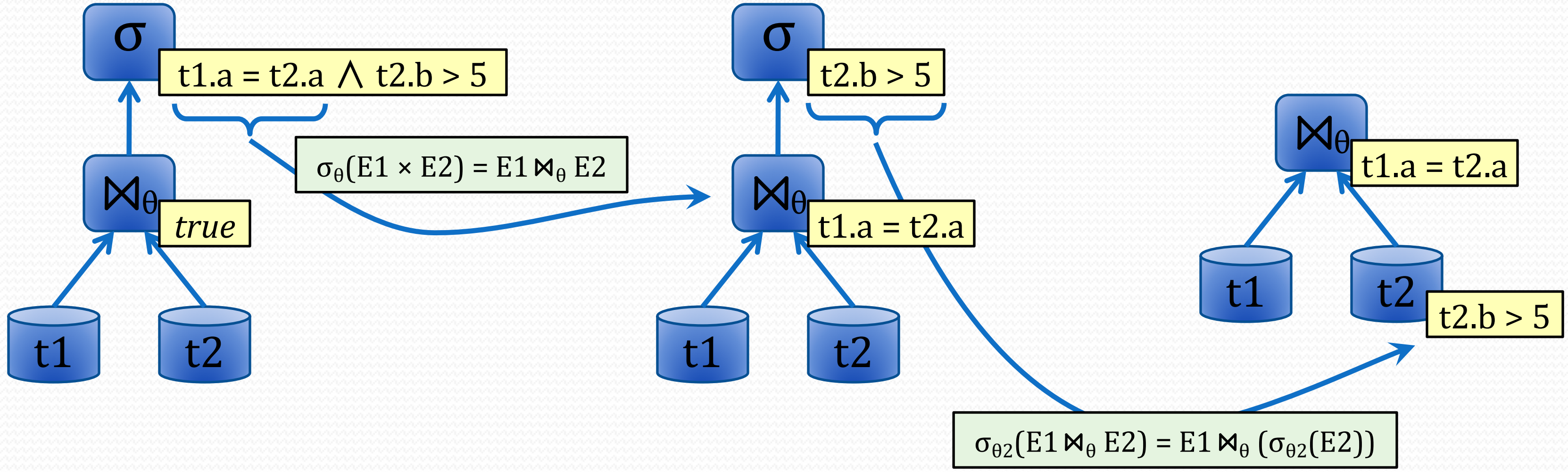
- Can sometimes distribute selects over theta-joins:
  - $\sigma_{\theta 1}(E1 \bowtie_{\theta} E2) = \sigma_{\theta 1}(E1) \bowtie_{\theta} E2$
    - $\theta 1$ only refers to attributes in E1
  - $\sigma_{\theta 1 \wedge \theta 2}(E1 \bowtie_{\theta} E2) = \sigma_{\theta 1}(E1) \bowtie_{\theta} \sigma_{\theta 2}(E2)$
    - $\theta 1$ only refers to attributes in E1
    - $\theta 2$ only refers to attributes in E2

Equivalence Rules

- Many other equivalence rules besides these
  - Cover grouping, projects, outer joins, set operations, duplicate elimination, sorting, etc.
- Grouping: $\sigma_\theta({}_A\mathcal{G}_F(E))$ is equivalent to ${}_A\mathcal{G}_F(\sigma_\theta(E))$
  - …as long as θ only involves attributes in A!
- Outer joins: $\sigma_\theta(E1 ⟕ E2)$ is equivalent to $\sigma_\theta(E1) ⟕ E2$
  - θ only involves attributes in E1

Equivalence Rules

- Equivalence rules allow us to transform plans, and know the results will not change:



Outer Join Transformations

- Need to be very careful transforming outer joins:
  - Obviously correct equivalences for natural joins / theta joins don't necessarily hold for outer joins!
- Is $\sigma_\theta(E1 ⟕ E2)$ equivalent to $E1 ⟕ \sigma_\theta(E2)$?
  - θ only uses attributes in E2
  - These are <u>not</u> equivalent. Example:
    - r(A, B) with one row { (1, 2) }
    - s(B, C) with one row { (2, 3) }
    - θ is C = 1
    - $\sigma_{C=1}(r ⟕ s)$ = { } *(empty relation)*, but $r ⟕ \sigma_{C=1}(s)$ = { (1, 2, *null*) }

Outer Join Transformations (2)

- Need to be very careful transforming outer joins:
  - Obviously correct equivalences for natural joins / theta joins don't necessarily hold for outer joins!
- Is (E1 ⟕ E2) ⟕ E3 equivalent to E1 ⟕ (E2 ⟕ E3)?
  - These are not equivalent. Example:
    - r(A, B) with one row { (1, 2) }
    - s(A, C) with one row { (2, 3) }
    - t(A, D) with one row { (1, 4) }
    - (r ⟕ s) ⟕ t = { (1, 2, *null*) } ⟕ t = { (1, 2, *null*, 4) }
    - r ⟕ (s ⟕ t) = r ⟕ { (2, 3, *null*) } = { (1, 2, *null*, *null*) }

Query Plan Optimization

- Generally understand how to map SQL queries to plans
  - Ignoring subqueries in SELECT and WHERE clauses for the time being…
- Understand how to implement basic plan nodes
  - Still a lot of optimizations to cover though…
- A query can be evaluated by many different plans…
- How do we find an *optimal* plan to evaluate a query?
  - Many different approaches
  - <u>All</u> depend on equivalence rules to guide generation of equivalent plans