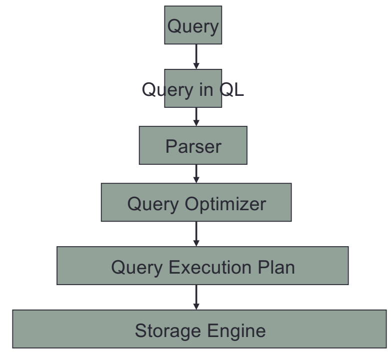
Query Optimization

- Query languages are declarative
- We need to
  - convert declarative statements into executable statements
  - estimate the cost of the executable statements
  - choose the cheapest executable statement among all alternatives

Query processing stages



Query Optimization

- In Traditional Databases, we are optimizing for cost of query processing
  - Cost is described in terms of disk access
  - Database keeps statistics about relations, tuples, page sizes.
  - Database also keeps index and sorting information
- Query optimizer
  - uses “cost model’ to guess query execution cost for a possible query execution plan
  - chooses a plan which is cheap according to the cost model.

Relational Data

- Data is Stored in “Tables”
- The size of each “row” in a table is the same.
- Columns can be
  - without index or
  - indexes (B^+- tree, Bitmap index, Hash index)
- Columns can be
  - non-clustered
  - clustered (easy access to close/some values)
  - sorted

SQL Query

SELECT $t_1.a$, $t_2.b$
(DISTINCT,SUM(),COUNT())
FROM TABLE1 t_1, TABLE2 t_2
WHERE $t_1.a = t_2.b$ AND $t_1.c > 10$
GROUP BY $t_1.a$

One Heuristic:

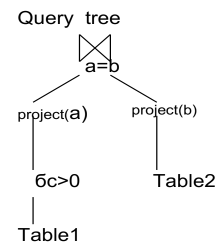
- Apply Π, б first
- Apply Joins Next
- Apply Group By Last

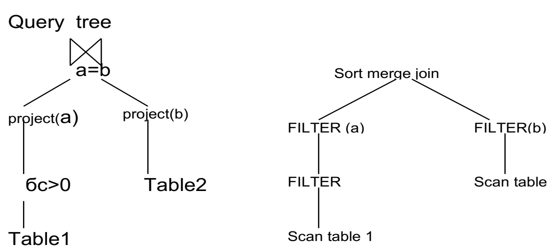
$\Pi_{t_1.a,\ t_2.b}\ [\sigma_{t1.c > 10}\ (\text{TABLE1} \bowtie_{t_1.a = t_2.b} \text{TABLE2})]$

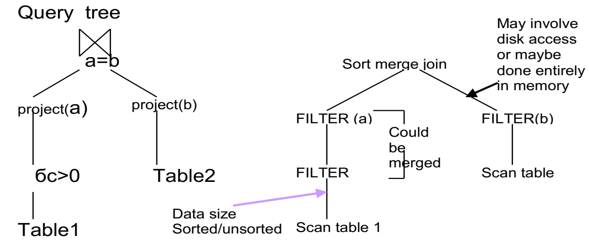
$\Pi_{t_1.a,\ t_2.b}\ [\sigma_{t1.a = t2.b,\ t1.c > 10}\ (\text{TABLE1 X TABLE2})]$

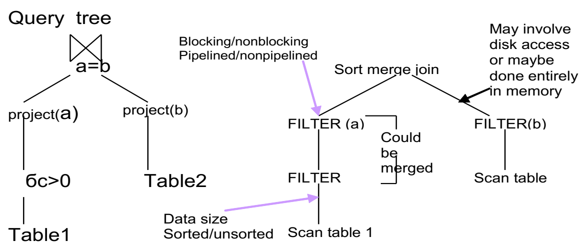
$\Pi_{t_1.a,\ t_2.b}\ [\sigma_{t1.c > 10}\ (\text{TABLE1} \bowtie_{t_1.a = t_2.b} \text{TABLE2})]$

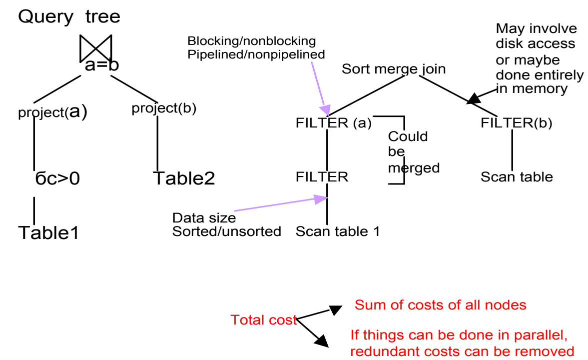
$\Pi_{t_1.a}\ [\sigma_{t1.c > 10}\ (\text{TABLE1})] \bowtie_{a = b} \Pi_{t_2.b}\ (\text{TABLE2})]$











Query Optimization

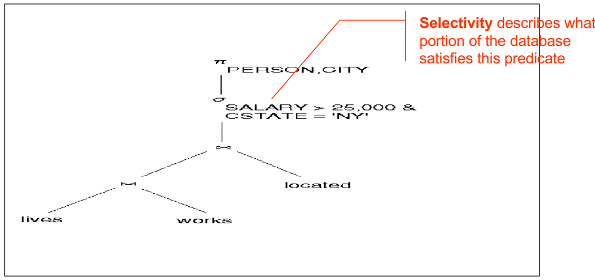
- Heuristic-based Query Optimization (CSE 412)
- Cost-based Query Optimization
  - System R Query Optimizer

Solution:

- Heuristics:
  - perform cheaper operations first
  - cheaper operations eliminate some of the inputs, hence more expensive operations need to deal with a smaller number of inputs.
- Challenge:
  - This assumes that we can freely change the order of operations (associatively, commutativity)

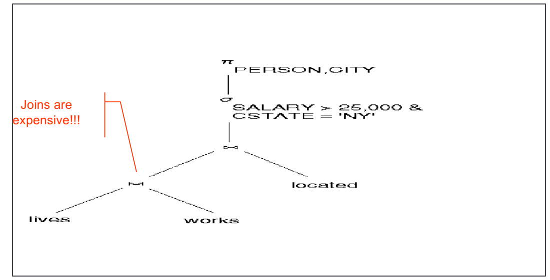
15

Query tree



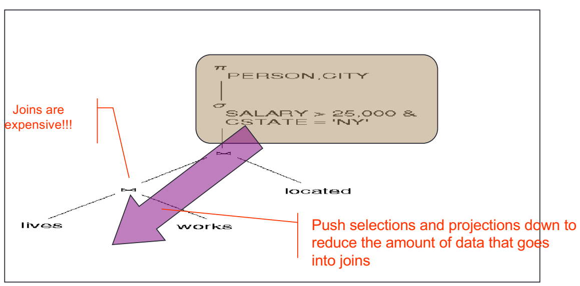
16

Query tree



17

Query tree



18

Cost-based Query Optimization

- Algebraic Transformations
  - commutative
  - distributive, etc.
- Operator Transformation
  - nested loop Vs sort merge join
- Operator insertion

- We are optimizing for cost of query processing
  - Cost is described in terms of disk access
  - Database keeps statistics about relations, tuples, page sizes.
  - Database also keeps index and sorting information
- Query optimizer uses "cost model' to guess query execution cost for a possible query execution plan
  - And chooses a plan which is least costly according to the cost model.

Cost based query optimization

- Cost Estimation
- Plan enumeration/search space generation
- Search algorithm

Problems

- What is a good cost model?
- What kind of statistics the database should keep?
- How should the query optimizer create alternative query execution plans?

Cost estimation

- What is the cost of reading one table from the disc?
  (SCAN)
- What is the cost of doing a selection?
  (FILTER)
- What is the cost of joining two tables?
  (JOIN)

Statistics:

- We need to estimate the number of disk I/Os.
  - Logical operators / physical operators
  - sizes of intermediate relations
  - ordering of operations (such as joins)
- How do we estimate cost of operations and sizes of resulting relations?
  - B(R) <- number of blocks
  - T(R) <- number of tuples
  - V(R.a) <- number of distinct values for an attribute "a"
  - V(R.[a_1----------a_n] <- number of distinct values when a_1----------a_n are considered together.

- Cost Estimates
  - I/O cost + CPU cost
- Scan Cost
  - B(R) (unsorted)
  - B(R) + #of index pages (sorted on a clustered index)
  - T(R) + # of index pages (sorted on a non-clustered index)
- (We already talked about the affects of database buffers)

Example use of statistics:

- Projection
  - easier
    - omit the attributes that are not included in the projection
    - recalculate how many blocks would be needed to store the resulting tuples.
  - What if distinct is specified?
- Selection
  - S = $\sigma_{a=c}$ (R) when "c" is a constant
  - T(S) = $\frac{T(R)}{V(R.a)}$
  - S = $\sigma_{a<>c}$ (R) when "c" is a constant
  - T(S) = $\frac{T(R)}{V(R.a)}$ [V(R.a)-1] or T(S) = T(R) ????

- S = $\sigma_{c1 \text{ or } c2}$ (R)
  - Assume c_1 and c_2 are independent
  - T(S) = T(R) * [1 - (1 - $\frac{n}{T(R)}$)* (1 - $\frac{m}{T(R)}$)],
    - where "n" is the number of tuples satisfying c_1, and "m" is the number of tuples satisfying c_2

34

- Join
  - $R(x,y) \bowtie S(y,z)$
    - $T(R \bowtie S) = 0$
    - $T(R \bowtie S) = T(R)$
    - $T(R \bowtie S) = T(R)T(S)$
  - $T(R \bowtie S) = \dfrac{T(R)T(S)}{\max(V(R,y),V(S,y))}$
  - $R(x,y_1, y_2) \bowtie S(y_1, y_2,z)$
  - $T(R \bowtie S) = \dfrac{T(R)T(S)}{\max(V(R,y_1)V(S, y_1))\max(V(R, y_2),V(S, y_2))}$
- Alternative: Keep Histograms!

35

- Join-cost
  - Nested Loop join
    $scan(R_1) + T(R_1) * \underline{scan(R_2)}$
  - Sort-merge join
    $sorted\text{-}scan(R_1) + sorted\text{-}scan(R_2)$

36

Assumptions of cost-based query optimization

- Independent of how a query is executed we get the same number of results:

$$\sigma_{\theta 1 \wedge \theta 2}(R) = \sigma_{\theta 1}(\sigma_{\theta 2}(R)) = \sigma_{\theta 2}(\sigma_{\theta 1}(R))$$

37

Assumptions of cost-based query optimization

- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries

Assumptions of cost-based query optimization

- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries

$$cost^1\left(\sigma_{\theta 1}\left(\sigma^1_{\theta 2}(R)\right)\right) = cost\left(\sigma^1_{\theta 2}(R)\right) + f\left(size\left(\sigma^1_{\theta 2}(R)\right), \theta 1\right)$$

$$cost^2\left(\sigma_{\theta 1}\left(\sigma^2_{\theta 2}(R)\right)\right) = cost\left(\sigma^2_{\theta 2}(R)\right) + f\left(size\left(\sigma^2_{\theta 2}(R)\right), \theta 1\right)$$

Assumptions of cost-based query optimization

- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries

$$cost^1\left(\sigma_{\theta 1}\left(\sigma^1_{\theta 2}(R)\right)\right) = cost\left(\sigma^1_{\theta 2}(R)\right) + f\left(size\left(\sigma^1_{\theta 2}(R)\right), \theta 1\right)$$

=

$$cost^2\left(\sigma_{\theta 1}\left(\sigma^2_{\theta 2}(R)\right)\right) = cost\left(\sigma^2_{\theta 2}(R)\right) + f\left(size\left(\sigma^2_{\theta 2}(R)\right), \theta 1\right)$$

Assumptions of cost-based query optimization

- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries

$$cost^1\left(\sigma_{\theta 1}\left(\sigma^1_{\theta 2}(R)\right)\right) = cost\left(\sigma^1_{\theta 2}(R)\right) + f\left(size\left(\sigma^1_{\theta 2}(R)\right), \theta 1\right)$$

=

$$cost^2\left(\sigma_{\theta 1}\left(\sigma^2_{\theta 2}(R)\right)\right) = cost\left(\sigma^2_{\theta 2}(R)\right) + f\left(size\left(\sigma^2_{\theta 2}(R)\right), \theta 1\right)$$

Assumptions of cost-based query optimization

- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries

$$cost^1\left(\sigma_{\theta 1}\left(\sigma^1_{\theta 2}(R)\right)\right) = cost\left(\sigma^1_{\theta 2}(R)\right) + f\left(size\left(\sigma^1_{\theta 2}(R)\right), \theta 1\right)$$

=

$$cost^2\left(\sigma_{\theta 1}\left(\sigma^2_{\theta 2}(R)\right)\right) = cost\left(\sigma^2_{\theta 2}(R)\right) + f\left(size\left(\sigma^2_{\theta 2}(R)\right), \theta 1\right)$$

Pick the cheapest!!!

Assumptions of cost-based query optimization

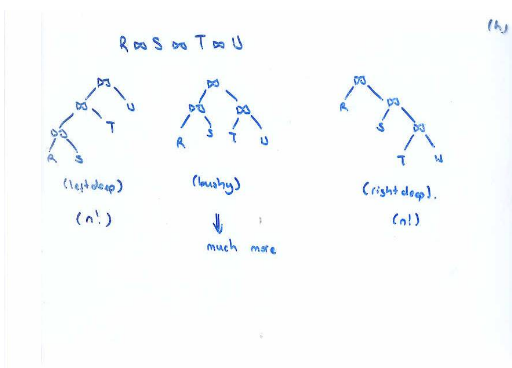
- Principle of sub-query optimality: An optimal plan for a query includes optimal subplans for the subqueries
- …we can use recursion!!!!!!!!!

Example:

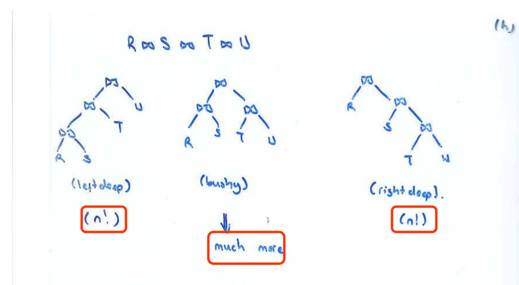
R_1 (a,b,c) ⋈ R_2 (a,d) ⋈ R_3 (d,e) ⋈ R_4 (c,e)

- Question: Which order of join is the best?
- Important:
  - The join ordering problem is NP-Hard!
    - I.e., there is no known polynomial time algorithm
    - There is strong reasons to believe that there is no polynomial time algorithm

What about join orders???



What about join orders ???



The goal of query optimization is to **eliminate costliest** executions….
……………………………………..finding cheapest executions is very very expensive.

Recursion

- Given Q = R1xR2xR3x….Rn
  - for all Ri,
    - find the cheapest plan for $Q_{(-Ri)}$
    - compute costi = cost($Q_{(-Ri)}$ x Ri)
  - Pick the plan with the smallest cost

Recursion

- Given Q = R1xR2xR3x….Rn
  - for all Ri,
    - find the cheapest plan for $Q_{(-Ri)}$
    - compute costi = cost($Q_{(-Ri)}$ x Ri)
  - Pick the plan with the smallest cost

Recursion!!!

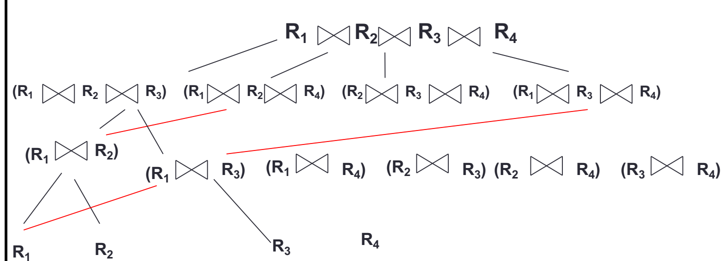
Recursion

- Given Q = R1xR2xR3x….Rn
  - for all Ri,
    - find the cheapest plan for $Q_{(-Ri)}$
    - compute costi = cost($Q_{(-Ri)}$ x Ri)
  - Pick the plan with the smallest cost

Recursion!!!

- There is a dynamic programming based algorithm that uses this recursion.

Dynamic Programming



53

Additional heuristics

- Avoid cartesian products
  - No index, hash, or sort-merge opportunity
  - The output is large
- Do not maintain one single plan per subexpression, but maintain multiple plans for "interesting" orders
  - If the result of the subexpression is sorted on the attributes specified in a sort operator in the query
  - If the attributes are used in a group_by operator
  - If they are used in a later join

54

Rule-based optimizers

- Starburst
  - Query is represented as a graph
  - If-C-then-A rules are used to apply algebraic transformations
  - A separate plan optimizer is used for physical query optimization

55

Rule-based optimizers

- Volcano
  - Query is represented as a tree
  - All transformations (logical, physical) treated uniformly
  - It uses a top-down query optimization algorithm (brach and bound)
    - Subexpressions are optimized on demand

56

Branch-and-bound

- Use heuristics to find a good physical plan
  - (let its cost be C)
- Refine the plan as follows
  - Consider other plans for subqueries
  - Any subquery with cost>C is ignored
  - If you find a complete plan with cost C' <C then replace the plan.

Greedy Algorithm

- Make a decision at a time about joins
  - Never backtrack

$$s = \frac{\text{size of the join result}}{\text{size of the current result}}$$

- Always pick the option with smallest s