Chapter 10: Query Optimization
Chapter 10: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Catalog Information for Cost Estimation
- Statistical Information for Cost Estimation
- Cost-based optimization
- Dynamic Programming for Choosing Evaluation Plans
Introduction

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
Introduction (Cont.)

- An evaluation plan defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.
Cost difference between evaluation plans for a query can be enormous
  - E.g. seconds vs. days in some cases

Steps in cost-based query optimization
  1. Generate logically equivalent expressions using equivalence rules
  2. Annotate resultant expressions to get alternative query plans
  3. Choose the cheapest plan based on estimated cost

Estimation of plan cost based on:
  - Statistical information about relations. Examples:
    - number of tuples, number of distinct values for an attribute
  - Statistics estimation for intermediate results
    - to compute cost of complex expressions
  - Cost formula for algorithms, computed using statistics
Generating Equivalent Expressions
Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every legal database instance.
  - Note: order of tuples is irrelevant.

- In SQL, inputs and outputs are multisets of tuples.
  - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance.

- An **equivalence rule** says that expressions of two forms are equivalent.
  - Can replace expression of first form by second, or vice versa.
Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.
   \[ \sigma_{\theta_1 \land \theta_2} (E) = \sigma_{\theta_1} (\sigma_{\theta_2} (E)) \]

2. Selection operations are commutative.
   \[ \sigma_{\theta_1} (\sigma_{\theta_2} (E)) = \sigma_{\theta_2} (\sigma_{\theta_1} (E)) \]

3. Only the last in a sequence of projection operations is needed, the others can be omitted.
   \[ \Pi_{L_1} (\Pi_{L_2} (\ldots (\Pi_{L_n} (E))\ldots)) = \Pi_{L_1} (E) \]

4. Selections can be combined with Cartesian products and theta joins.
   a. \[ \sigma_{\theta} (E_1 \times E_2) = E_1 \Join_{\theta} E_2 \]
   b. \[ \sigma_{\theta_1} (E_1 \Join_{\theta_2} E_2) = E_1 \Join_{\theta_1 \land \theta_2} E_2 \]
5. Theta-join operations (and natural joins) are commutative.

\[ E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1 \]

6. (a) Natural join operations are associative:

\[ (E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3) \]

(b) Theta joins are associative in the following manner:

\[ (E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \land \theta_3} E_3 = E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_2} E_3) \]

where \( \theta_2 \) involves attributes from only \( E_2 \) and \( E_3 \).
Pictorial Depiction of Equivalence Rules

Rule 5

Rule 6a

Rule 7a
If $\theta$ only has attributes from E1
7. The selection operation distributes over the theta join operation under the following two conditions:
(a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions ($E_1$) being joined.

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

(b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

$$\sigma_{\theta_1 \land \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$
8. The projection operation distributes over the theta join operation as follows:

(a) if $\theta$ involves only attributes from $L_1 \cup L_2$:

$$
\Pi_{L_1 \cup L_2} (E_1 \Join_\theta E_2) = (\Pi_{L_1} (E_1)) \Join_\theta (\Pi_{L_2} (E_2))
$$

(b) Consider a join $E_1 \Join_\theta E_2$.

- Let $L_1$ and $L_2$ be sets of attributes from $E_1$ and $E_2$, respectively.
- Let $L_3$ be attributes of $E_1$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$, and
- let $L_4$ be attributes of $E_2$ that are involved in join condition $\theta$, but are not in $L_1 \cup L_2$.

$$
\Pi_{L_1 \cup L_2} (E_1 \Join_\theta E_2) = \Pi_{L_1 \cup L_2} ((\Pi_{L_1 \cup L_3} (E_1)) \Join_\theta (\Pi_{L_2 \cup L_4} (E_2)))
$$
9. The set operations union and intersection are commutative

\[ E_1 \cup E_2 = E_2 \cup E_1 \]
\[ E_1 \cap E_2 = E_2 \cap E_1 \]

(set difference is not commutative).

10. Set union and intersection are associative.

\[(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)\]
\[(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)\]

11. The selection operation distributes over \( \cup \), \( \cap \) and \( - \).

\[ \sigma_{\theta} (E_1 - E_2) = \sigma_{\theta} (E_1) - \sigma_{\theta}(E_2) \]
and similarly for \( \cup \) and \( \cap \) in place of \( - \)

Also:
\[ \sigma_{\theta} (E_1 - E_2) = \sigma_{\theta}(E_1) - E_2 \]
and similarly for \( \cap \) in place of \( - \), but not for \( \cup \)

12. The projection operation distributes over union

\[ \Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2)) \]
Transformation Example: Pushing Selections

• Query: Find the names of all customers who have an account at some branch located in Brooklyn.
\[ \Pi_{customer\_name}(\sigma_{branch\_city = "Brooklyn"} (branch \Join (account \Join depositor))) \]

• Transformation using rule 7a.
\[ \Pi_{customer\_name}((\sigma_{branch\_city = "Brooklyn"} (branch)) \Join (account \Join depositor)) \]

• Performing the selection as early as possible reduces the size of the relation to be joined.
Example with Multiple Transformations

- Query: Find the names of all customers with an account at a Brooklyn branch whose account balance is over $1000.

\[
\prod_{\text{customer\_name}} (\sigma_{\text{branch\_city} = \text{“Brooklyn”} \land \text{balance} > 1000} (\text{branch} \ Join (\text{account} \ Join \text{depositor})))
\]

- Transformation using join associatively (Rule 6a):

\[
\prod_{\text{customer\_name}} (\sigma_{\text{branch\_city} = \text{“Brooklyn”} \land \text{balance} > 1000} (\text{branch} \ Join \text{account}) \ Join \text{depositor})
\]

- Second form provides an opportunity to apply the “perform selections early” rule, resulting in the subexpression

\[
\sigma_{\text{branch\_city} = \text{“Brooklyn”}} (\text{branch}) \ Join \sigma_{\text{balance} > 1000} (\text{account})
\]

- Thus a sequence of transformations can be useful
Multiple Transformations (Cont.)

(a) Initial expression tree

\[
\Pi_{\text{customer\_name}}
\sigma_{\text{branch\_city}=\text{Brooklyn}}
\land \text{balance} < 1000
\]

\[
\begin{array}{c}
\text{branch} \\
\text{account} \\
\text{depositor}
\end{array}
\]

(b) Tree after multiple transformations

\[
\Pi_{\text{customer\_name}}
\sigma_{\text{branch\_city}=\text{Brooklyn}}
\sigma_{\text{balance} < 1000}
\]

\[
\begin{array}{c}
\text{branch} \\
\text{account}
\end{array}
\]

\textit{depositor}
Transformation Example: Pushing Projections

\[ \Pi_{\text{customer\_name}}((\sigma_{\text{branch\_city} = \text{“Brooklyn”}} (\text{branch}) \bowtie \text{account}) \bowtie \text{depositor}) \]

- When we compute

\[ (\sigma_{\text{branch\_city} = \text{“Brooklyn”}} (\text{branch}) \bowtie \text{account} ) \]

we obtain a relation whose schema is:

\[ (\text{branch\_name}, \text{branch\_city}, \text{assets}, \text{account\_number}, \text{balance}) \]

- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:

\[ \Pi_{\text{customer\_name}}((\Pi_{\text{account\_number}}((\sigma_{\text{branch\_city} = \text{“Brooklyn”}} (\text{branch}) \bowtie \text{account}) \bowtie \text{depositor})) \bowtie \text{depositor}) \]

- Performing the projection as early as possible reduces the size of the relation to be joined.
Join Ordering Example

- For all relations \( r_1, r_2, \) and \( r_3, \)

\[
(r_1 \Join r_2) \Join r_3 = r_1 \Join (r_2 \Join r_3)
\]

(Join Associativity)

- If \( r_2 \Join r_3 \) is quite large and \( r_1 \Join r_2 \) is small, we choose

\[
(r_1 \Join r_2) \Join r_3
\]

so that we compute and store a smaller temporary relation.
Join Ordering Example (Cont.)

• Consider the expression

\[ \Pi_{\text{customer\_name}} ((\sigma_{\text{branch\_city} = \text{"Brooklyn"}} (\text{branch})) \bowtie (\text{account} \bowtie \text{depositor})) \]

• Could compute \( \text{account} \bowtie \text{depositor} \) first, and join result with \( \sigma_{\text{branch\_city} = \text{"Brooklyn"}} (\text{branch}) \) but \( \text{account} \bowtie \text{depositor} \) is likely to be a large relation.

• Only a small fraction of the bank’s customers are likely to have accounts in branches located in Brooklyn
  • it is better to compute

\[ \sigma_{\text{branch\_city} = \text{"Brooklyn"}} (\text{branch}) \bowtie \text{account} \]

first.
Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression.
- Can generate all equivalent expressions as follows:
  - Repeat
    - apply all applicable equivalence rules on every equivalent expression found so far
    - add newly generated expressions to the set of equivalent expressions
  Until no new equivalent expressions are generated above
- The above approach is very expensive in space and time
Cost Estimation

- Cost of each operator computer as described in Chapter 9
  - Need statistics of input relations
    - E.g. number of tuples, sizes of tuples
  - Inputs can be results of sub-expressions
    - Need to estimate statistics of expression results
    - To do so, we require additional statistics
      - E.g. number of distinct values for an attribute
- More on cost estimation later
Choice of Evaluation Plans

- Must consider the interaction of evaluation techniques when choosing evaluation plans
  - choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. E.g.
    - merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for an outer level aggregation.
    - nested-loop join may provide opportunity for pipelining
  - Practical query optimizers incorporate elements of the following two broad approaches:
    1. Search all the plans and choose the best plan in a cost-based fashion.
    2. Uses heuristics to choose a plan.
Cost-Based Optimization

- Consider finding the best join-order for \( r_1 \bowtie r_2 \bowtie \ldots r_n \).
- There are \((2(n-1))!/(n-1)!\) different join orders for above expression. With \( n = 7 \), the number is 665280, with \( n = 10 \), the number is greater than 176 billion!
- No need to generate all the join orders. Using dynamic programming, the least-cost join order for any subset of \( \{r_1, r_2, \ldots r_n\} \) is computed only once and stored for future use.
Dynamic Programming in Optimization

- To find best join tree for a set of $n$ relations:
  - To find best plan for a set $S$ of $n$ relations, consider all possible plans of the form: $S_1 \Join (S - S_1)$ where $S_1$ is any non-empty subset of $S$.
  - Recursively compute costs for joining subsets of $S$ to find the cost of each plan. Choose the cheapest of the $2^n - 1$ alternatives.
- Base case for recursion: single relation access plan
  - Apply all selections on $R_i$ using best choice of indices on $R_i$
  - When plan for any subset is computed, store it and reuse it when it is required again, instead of recomputing it
    - Dynamic programming
procedure findbestplan(S)
    if (bestplan[S].cost ≠ ∞)
        return bestplan[S]
    // else bestplan[S] has not been computed earlier, compute it now
    if (S contains only 1 relation)
        set bestplan[S].plan and bestplan[S].cost based on the best way of accessing S /* Using selections on S and indices on S */
    else for each non-empty subset S1 of S such that S1 ≠ S
        P1 = findbestplan(S1)
        P2 = findbestplan(S - S1)
        A = best algorithm for joining results of P1 and P2
        cost = P1.cost + P2.cost + cost of A
        if cost < bestplan[S].cost
            bestplan[S].cost = cost
            bestplan[S].plan = “execute P1.plan; execute P2.plan; join results of P1 and P2 using A”
    return bestplan[S]
Cost of Optimization

- With dynamic programming time complexity of optimization with bushy trees is $O(3^n)$.
  - With $n = 10$, this number is 59000 instead of 176 billion!
- Space complexity is $O(2^n)$
- To find best left-deep join tree for a set of $n$ relations:
  - Consider $n$ alternatives with one relation as right-hand side input and the other relations as left-hand side input.
  - Modify optimization algorithm:
    - Replace "for each non-empty subset $S1$ of $S$ such that $S1 \neq S$"
    - By: for each relation $r$ in $S$
      let $S1 = S – r$
  - If only left-deep trees are considered, time complexity of finding best join order is $O(n 2^n)$
    - Space complexity remains at $O(2^n)$
- Cost-based optimization is expensive, but worthwhile for queries on large datasets (typical queries have small $n$, generally $< 10$)
Heuristic Optimization

- Cost-based optimization is expensive, even with dynamic programming.
- Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improve execution performance:
  - Perform selection early (reduces the number of tuples)
  - Perform projection early (reduces the number of attributes)
  - Perform most restrictive selection and join operations (i.e. with smallest result size) before other similar operations.
  - Some systems use only heuristics, others combine heuristics with partial cost-based optimization.
Statistics for Cost Estimation
Statistical Information for Cost Estimation

- $n_r$: 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 $\prod_A(r)$.
- If tuples of $r$ are stored together physically in a file, then:

$$b_r = \left\lfloor \frac{n_r}{f_r} \right\rfloor$$
Histograms

- Histogram on attribute *age* of relation *person*

![Histogram of age distribution over different ranges](image)

- Equi-width histograms
- Equi-depth histograms
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 $n_r / 2$. 
The selectivity of a condition $\theta_i$ is the probability that a tuple in the relation $r$ satisfies $\theta_i$.

- If $s_i$ is the number of satisfying tuples in $r$, the selectivity of $\theta_i$ is given by $s_i / n_r$.

**Conjunction**: $\sigma_{\theta_1 \land \theta_2 \land \ldots \land \theta_n}(r)$. Assuming independence, estimate of tuples in the result is:

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

**Disjunction**: $\sigma_{\theta_1 \lor \theta_2 \lor \ldots \lor \theta_n}(r)$. Estimated number of tuples:

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

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

$$n_r - \text{size}(\sigma_{\theta}(r))$$
Join Operation: Running Example

Running example:

\( \text{depositor} \bowtie \text{customer} \)

Catalog information for join examples:

- \( n_{\text{customer}} = 10,000 \).
- \( f_{\text{customer}} = 25 \), which implies that \( b_{\text{customer}} = 10000/25 = 400 \).
- \( n_{\text{depositor}} = 5000 \).
- \( f_{\text{depositor}} = 50 \), which implies that \( b_{\text{depositor}} = 5000/50 = 100 \).
- \( V(\text{customer\_name}, \text{depositor}) = 2500 \), which implies that, on average, each customer has two accounts.
  - Also assume that \( \text{customer\_name} \) in \( \text{depositor} \) is a foreign key on \( \text{customer} \).
- \( V(\text{customer\_name}, \text{customer}) = 10000 \) (primary key!)
Estimation of the Size of Joins

- The Cartesian product \( r \times s \) contains \( n_r \cdot n_s \) tuples; each tuple occupies \( s_r + s_s \) bytes.

- If \( R \cap S = \emptyset \), then \( r \bowtie s \) is the same as \( r \times s \).

- If \( R \cap S \) is a key for \( R \), then a tuple of \( s \) will join with at most one tuple from \( r \)
  - therefore, the number of tuples in \( r \bowtie s \) is no greater than the number of tuples in \( s \).

- If \( R \cap S \) in \( S \) is a foreign key in \( S \) referencing \( R \), then the number of tuples in \( r \bowtie s \) is exactly the same as the number of tuples in \( s \).
  - The case for \( R \cap S \) being a foreign key referencing \( S \) is symmetric.

- In the example query \( \text{depositor} \bowtie \text{customer}, \text{customer\_name} \) in \( \text{depositor} \) is a foreign key of \( \text{customer} \)
  - hence, the result has exactly \( n_{\text{depositor}} \) tuples, which is 5000
Estimation of the Size of Joins (Cont.)

- If \( R \cap S = \{A\} \) is not a key for \( R \) or \( S \).
  If we assume that every tuple \( t \) in \( R \) produces tuples in \( R \times S \), the number of tuples in \( R \times S \) is estimated to be:
  \[
  \frac{n_r \times n_s}{V(A,s)}
  \]
  If the reverse is true, the estimate obtained will be:
  \[
  \frac{n_r \times n_s}{V(A,r)}
  \]
  The lower of these two estimates is probably the more accurate one.

- Can improve on above if histograms are available
  - Use formula similar to above, for each cell of histograms on the two relations
Estimation of the Size of Joins (Cont.)

- Compute the size estimates for \textit{depositor} \times \textit{customer} without using information about foreign keys:
  - \( V(\text{customer\_name, depositor}) = 2500 \), and \( V(\text{customer\_name, customer}) = 10000 \)
  - The two estimates are \( 5000 \times 10000/2500 - 20,000 \) and \( 5000 \times 10000/10000 = 5000 \)
  - We choose the lower estimate, which in this case, is the same as our earlier computation using foreign keys.
Selections: $\sigma_\theta (r)$

- If $\theta$ forces $A$ to take a specified value: $V(A, \sigma_\theta (r)) = 1$.
  - e.g., $A = 3$

- If $\theta$ forces $A$ to take on one of a specified set of values: $V(A, \sigma_\theta (r)) = \text{number of specified values}$.
  - (e.g., ($A = 1 \ V A = 3 \ V A = 4$)),

- If the selection condition $\theta$ is of the form $A \ op \ r$
  
  estimated $V(A, \sigma_\theta (r)) = V(A.r) * s$
  
  - where $s$ is the selectivity of the selection.

- In all the other cases: use approximate estimate of $\min(V(A,r), n_{\sigma_\theta (r)})$
  
  - More accurate estimate can be got using probability theory, but this one works fine generally
Estimation of Distinct Values (Cont.)

Joins: $r \bowtie s$

- If all attributes in $A$ are from $r$
  
  \[
  \text{estimated } V(A, r \bowtie s) = \min (V(A,r), n_r \bowtie s)
  \]

- If $A$ contains attributes $A_1$ from $r$ and $A_2$ from $s$, then
  
  \[
  V(A, r \bowtie s) = \min(V(A_1, r)*V(A_2 - A_1, s), V(A_1 - A_2, r)*V(A_2, s), n_r \bowtie s)
  \]

- More accurate estimate can be got using probability theory, but this one works fine generally
Estimation of Distinct Values (Cont.)

- Estimation of distinct values are straightforward for projections.
  - They are the same in $\Pi_A(r)$ as in $r$.
- The same holds for grouping attributes of aggregation.
- For aggregated values
  - For min($A$) and max($A$), the number of distinct values can be estimated as $\min(V(A,r), V(G,r))$ where $G$ denotes grouping attributes
  - For other aggregates, assume all values are distinct, and use $V(G,r)$
End of Chapter 10