Chapter 13: Query Optimization
Chapter 13: 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
An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.

\[
\Pi_{\text{name, title}} \text{ (sort to remove duplicates)}
\]

\[
\text{(hash join)}
\]

\[
\text{(merge join)}
\]

\[
\sigma_{\text{dept\_name} = \text{Music}} \text{ (use index 1)}
\]

\[
\sigma_{\text{year} = 2009} \text{ (use linear scan)}
\]

\[
\text{course}
\]

\[
\text{teaches}
\]

\[
\text{instructor}
\]
Introduction (Cont.)

- 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 formulae 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.
  - We don’t care if they generate different results on databases that violate integrity constraints.
- 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. \[ \sigma_\theta (E_1 \times E_2) = E_1 \bowtie_\theta E_2 \]
   b. \[ \sigma_{\theta_1} (E_1 \bowtie \theta_2 E_2) = E_1 \bowtie_{\theta_1 \land \theta_2} E_2 \]
5. Theta-join operations (and natural joins) are commutative.
\[ E_1 \Join_{\theta} E_2 = E_2 \Join_{\theta} E_1 \]

6. (a) Natural join operations are associative:
\[ (E_1 \Join E_2) \Join E_3 = E_1 \Join (E_2 \Join E_3) \]

(b) Theta joins are associative in the following manner:
\[ (E_1 \Join_{\theta_1} E_2) \Join_{\theta_2 \land \theta_3} E_3 = E_1 \Join_{\theta_1 \land \theta_3} (E_2 \Join_{\theta_2} E_3) \]

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

Rule 5

E1 \theta E2

Rule 6a

E2 \theta E1

E2 \theta E1

E2 \theta E3

Rule 7a

\sigma_\theta

If \theta only has attributes from E1

E1 \sigma_\theta E2

E1 \sigma_\theta E2

E1 \sigma_\theta E2
Equivalence Rules (Cont.)

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 $E_2$.

   $$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$
Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

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

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

(b) Consider a join $E_1 \bowtie_{\theta} E_2$.

- 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 $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2} (E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2} (\Pi_{L_1 \cup L_3} (E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4} (E_2))$$
Equivalence Rules (Cont.)

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 \(\setminus\).
    \[ \sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta (E_2) \]
    and similarly for \(\cup\) and \(\cap\) in place of \(\setminus\).
    Also:
    \[ \sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - E_2 \]
    and similarly for \(\cap\) in place of \(\setminus\), 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 instructors in the Music department, along with the titles of the courses that they teach

\[ \Pi_{\text{name, title}}(\sigma_{\text{dept\_name}= \text{"Music"}}(\text{instructor} \bowtie (\text{teaches} \bowtie \Pi_{\text{course\_id, title}}(\text{course}))))) \]

■ Transformation using rule 7a.

\[ \Pi_{\text{name, title}}((\sigma_{\text{dept\_name}= \text{"Music"}}(\text{instructor})) \bowtie (\text{teaches} \bowtie \Pi_{\text{course\_id, title}}(\text{course})))) \]

■ 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 instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught

  - $\Pi_{name, \text{title}} (\sigma_{dept\_name = \text{"Music"} \land year = 2009} (instructor \bowtie (teaches \bowtie \Pi_{\text{course\_id, title}} (course)))))$

- Transformation using join associatively (Rule 6a):

  - $\Pi_{name, \text{title}} (\sigma_{dept\_name = \text{"Music"} \land year = 2009} ((instructor \bowtie teaches) \bowtie \Pi_{\text{course\_id, title}} (course))))$

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

  $\sigma_{dept\_name = \text{"Music"}} (instructor) \bowtie \sigma_{year = 2009} (teaches)$
Multiple Transformations (Cont.)

\[ \Pi_{\text{name, title}} \]
\[ \sigma_{\text{dept\_name} = \text{Music} \land \text{year} = 2009} \]

\[ \text{instructor} \]
\[ \text{teaches} \]
\[ \text{course} \]

(a) Initial expression tree

\[ \Pi_{\text{name, title}} \]
\[ \sigma_{\text{dept\_name} = \text{Music}} \]
\[ \sigma_{\text{year} = 2009} \]

\[ \text{instructor} \]
\[ \text{teaches} \]
\[ \text{course} \]

(b) Tree after multiple transformations
Transformation Example: Pushing Projections

- Consider: \( \Pi_{\text{name}, \text{title}} (\sigma_{\text{dept_name}=\text{"Music"}} (\text{instructor}) \bowtie \text{teaches}) \bowtie \Pi_{\text{course_id}, \text{title}} (\text{course})) \)

- When we compute
  \( (\sigma_{\text{dept_name}=\text{"Music"}} (\text{instructor}) \bowtie \text{teaches}) \)
  we obtain a relation whose schema is:
  \( (\text{ID}, \text{name}, \text{dept_name}, \text{salary}, \text{course_id}, \text{sec_id}, \text{semester}, \text{year}) \)

- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get:
  \( \Pi_{\text{name}, \text{title}} (\Pi_{\text{name}, \text{course_id}} (\sigma_{\text{dept_name}=\text{"Music"}} (\text{instructor}) \bowtie \text{teaches}) \bowtie \Pi_{\text{course_id}, \text{title}} (\text{course}))) \)

- 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

\[
\Pi_{\text{name, title}}(\sigma_{\text{dept_name}=\text{"Music"}}(\text{instructor}) \bowtie \text{teaches}) \bowtie \Pi_{\text{course_id, title}}(\text{course}))
\]

- Could compute \( \text{teaches} \bowtie \Pi_{\text{course_id, title}}(\text{course}) \) first, and join result with

\[
\sigma_{\text{dept_name}=\text{"Music"}}(\text{instructor})
\]

but the result of the first join is likely to be a large relation.

- Only a small fraction of the university’s instructors are likely to be from the Music department

  - it is better to compute

\[
\sigma_{\text{dept_name}=\text{"Music"}}(\text{instructor}) \bowtie \text{teaches}
\]

  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 subexpression of 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.
  - Two approaches
    - Optimized plan generation based on transformation rules
    - Special case approach for queries with only selections, projections and joins
Cost Estimation

- Cost of each operator computed as described in previous set of slides
  - 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 $r_1 \times r_2 \times \ldots \times r_n$.
- 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 $2^n - 2$ 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
Interesting Sort Orders

- Consider the expression \((r_1 \bowtie r_2) \bowtie r_3\) (with A as common attribute)

- An **interesting sort order** is a particular sort order of tuples that could be useful for a later operation
  - Using merge-join to compute \(r_1 \bowtie r_2\) may be costlier than hash join but generates result sorted on A
  - Which in turn may make merge-join with \(r_3\) cheaper, which may reduce cost of join with \(r_3\) and minimizing overall cost
  - Sort order may also be useful for order by and for grouping

- Not sufficient to find the best join order for each subset of the set of \(n\) given relations
  - must find the best join order for each subset, **for each interesting sort order**
  - Simple extension of earlier dynamic programming algorithms
  - Usually, number of interesting orders is quite small and doesn’t affect time/space complexity significantly
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:

$$b_r = \frac{n_r}{f_r}$$
Histograms

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

- **Equi-width** histograms
- **Equi-depth** histograms