Query Optimization

ULB

System R paper

System R: Relational Approach to Database Management

M. M. ASTRAHAN, M. W. BLASGEN, D. D. CHAMBERLIN,
K. P. ESWARAN, J. N. GRAY, P. P. GRIFFITHS,
W. F. KING, R. A. LORIE, P. R. McJONES, J. W. MEHL,
G. R. PUTZOLU, I. L. TRAIGER, B. W. WADE, AND V. WATSON

IBM Research Laboratory

ULB

System R paper - metadata

- Coming from IBM

- Published in ACM TODS

- Authors including:

  - Jim Gray - Turing award 1998, and others

  - Bruce G. Lindsay - ACM SIGMOD Edgar F. Codd Innovations Award, 2012, and others

  - Patricia G. Selinger - SIGMOD Edgar F. Codd Innovations Award, and others

ULB

System R paper - Discussion

1. What are the architecture components of system R ?

2. Which language did system R use for querying ?

3. What is a catalogue ?

4. What is a cursor ? Is it still used ?

5. What is a clustering image ?

6. How did the system R optimizer work ?

7. What are the parameters of cost estimation ?

8. What is an access path ?

ULB

Query Optimization

Alternative ways of evaluating a given query

- Equivalent expressions   RA
- Different algorithms for each operation

Logical execution plan
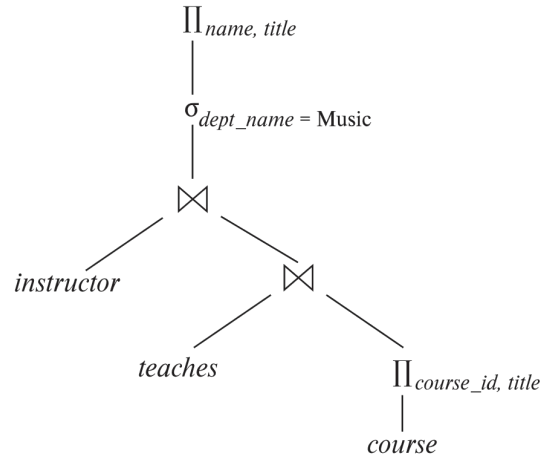
(a) Initial expression tree
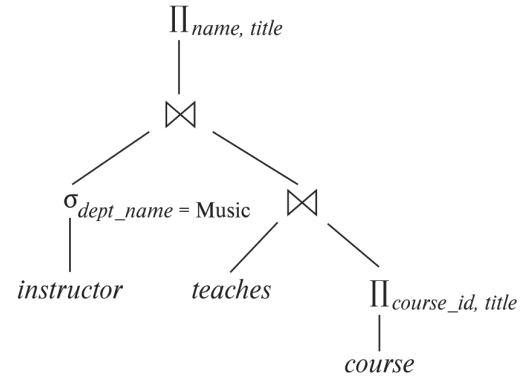
(b) Transformed expression tree

Query Plan

An evaluation plan defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.

$\Pi_{name,\ title}$ (sort to remove duplicates)

⋈ (merge join)

$sort_{ID}$

$sort_{ID}$

⋈ (hash join)

$\sigma_{dept\_name\ =\ Music}$ (use index 1)

$\Pi_{course\_id,\ title}$

instructor

teaches

course
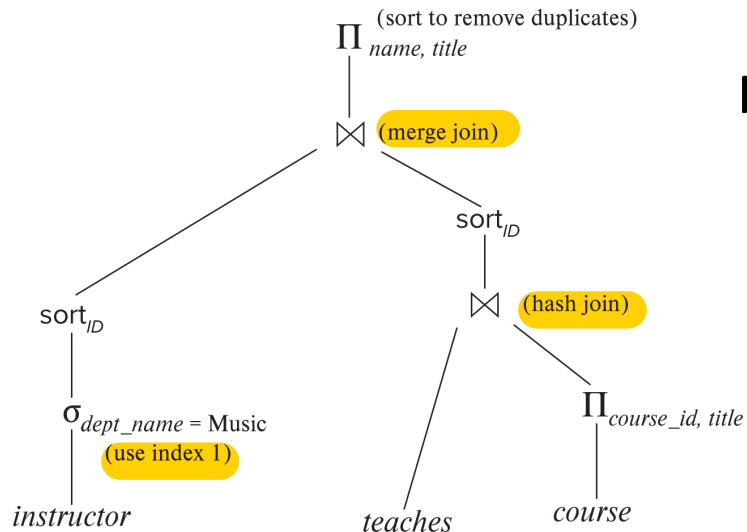
Physical execution plan

Find out how to view query execution plans on your favorite database

Cost-based Query Optimization

- 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
  a. Generate logically equivalent expressions using equivalence rules
  b. Annotate resultant expressions to get alternative query plans
  c. 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

Viewing Query Evaluation Plans

- Most database support explain <query>
    - Displays plan chosen by query optimizer, along with cost estimates
    - Some syntax variations between databases
        - Oracle: **explain plan for <query>** followed by **select * from table** (dbms_xplan.display)
        - SQL Server: set showplan_text on
- Some databases (e.g. PostgreSQL) support explain analyse <query>
    - Shows actual runtime statistics found by running the query, in addition to showing the plan
- Some databases (e.g. PostgreSQL) show cost as *f..l*
    - *f* is the cost of delivering first tuple and *l* is cost of delivering all results

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 (bag) 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, so one can replace the other

Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta 1 \wedge \theta 2}(E) \equiv \sigma_{\theta 1}(\sigma_{\theta 2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta 1}(\sigma_{\theta 2}(E)) \equiv \sigma_{\theta 2}(\sigma_{\theta 1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted (where L1 $\subseteq$ L2 ... $\subseteq$ Ln)

$$\prod_{L1}(\prod_{L2}(\ldots(\prod_{Ln}(E))\ldots)) \equiv \prod L_1(E)$$

4. Selections can be combined with Cartesian products and theta joins.

$$\sigma_\theta(E_1 \times E_2) \equiv E_1 \bowtie_\theta E_2$$

$$\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) \equiv E_1 \bowtie_{\theta 1 \wedge \theta 2} E$$

Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie E_2 \equiv E_2 \bowtie E_1$$

6. (a) Natural join operations are associative:
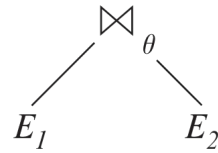
$$(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$$

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

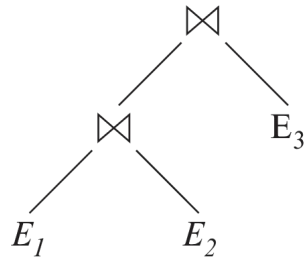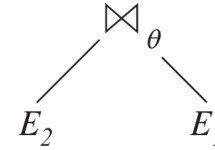$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 \equiv E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

where θ_2 involves attributes from only E_2 and E_3

Pictorial Depiction of Equivalence Rules
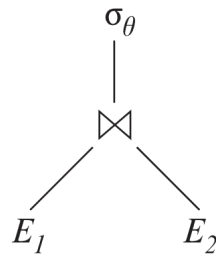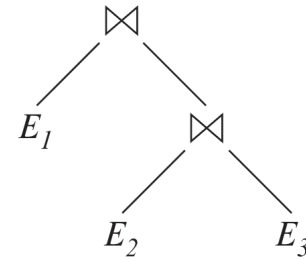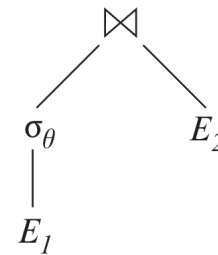
Equivalence Rules (Cont.)

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 (E1) being joined.

$$\sigma_{\theta 0} (E_1 \bowtie_\theta E_2) \equiv (\sigma_{\theta 0}(E_1)) \bowtie_\theta E_2$$

    (b) When $\theta 1$ involves only the attributes of E1 and $\theta 2$ involves only the attributes of E2.

$$\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_\theta E_2) \equiv (\sigma_{\theta 1}(E_1)) \bowtie_\theta (\sigma_{\theta 2}(E2))$$

Equivalence Rules (Cont.)

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

    (a) if θ involves only attributes from L1 ∪ L2:

    $$\prod_{L1 \cup L2}(E_1 \bowtie_\theta E_2) \equiv \prod_{L1}(E_1) \bowtie_\theta \prod_{L2}(E_2)$$

Similar equivalence hold for outerjoin operations: ⟕, ⟖, and ⟗

Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 \equiv E_2 \cup E_1 \qquad E_1 \cap E_2 \equiv E_2 \cap E_1$$

(set difference is not commutative).

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 \equiv E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 \equiv E_1 \cap (E_2 \cap E_3)$$

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

$$\sigma_\theta (E_1 \cup E_2) \equiv \sigma_\theta (E_1) \cup \sigma_\theta(E_2)$$
$$\sigma_\theta (E_1 \cap E_2) \equiv \sigma_\theta (E_1) \cap \sigma_\theta(E_2)$$
$$\sigma_\theta (E_1 - E_2) \equiv \sigma_\theta (E_1) - \sigma_\theta(E_2)$$
$$\sigma_\theta (E_1 \cap E_2) \equiv \sigma_\theta(E_1) \cap E_2$$
$$\sigma_\theta (E_1 - E_2) \equiv \sigma_\theta(E_1) - E_2 \text{ (does not hold for } \cup)$$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) \equiv (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Exercise

- Create equivalence rules involving
  - The group by/aggregation operation
  - Left outer join operation

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

$$\Pi_{name,\ title}(\sigma_{dept\_name=\ 'Music'}(instructor \bowtie (teaches \bowtie \Pi_{course\_id,\ title} (course))))$$

Transformation using rule 7a.

$$\Pi_{name,\ title}((\sigma_{dept\_name=\ 'Music'}(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,\ title} (course)))$$

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 2017, along with the titles of the courses that they taught

$$\Pi_{name, title}(\sigma_{dept\_name= \text{ "Music"} \wedge year=2017}$$
$$(instructor \bowtie (teaches \bowtie \Pi_{course\_id, title} (course))))$$

Transformation using join associatively (Rule 6a):

$$\Pi_{name, title}(\sigma_{dept\_name= \text{ "Music"} \wedge year = 2017}$$
$$((instructor \bowtie teaches) \bowtie \Pi_{course\_id, title} (course)))$$

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

$$\sigma_{dept\_name = \text{ "Music"}} (instructor) \bowtie \sigma_{year = 2017} (teaches)$$

Transformation Example: Pushing Projections

Consider:

$$\Pi_{name,title}(\sigma_{dept\_name=\text{“Music”}}(instructor)\bowtie teaches)\bowtie\Pi_{course\_id,title}(course))))$$

When we compute

$$(\sigma_{dept\_name=\text{“Music”}}(instructor \bowtie teaches)$$

we obtain a relation whose schema is:
   (ID, name, dept_name, salary, course_id, sec_id, semester, year)

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

$$\Pi_{name,title}(\Pi_{name,course\_id}(\sigma_{dept\_name=\text{“Music”}}(instructor) \bowtie teaches))$$
$$\bowtie \quad \Pi_{course\_id,\ title}(course))))$$

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,

$$(r_1 \bowtie r_2) \bowtie r_3 \;=\; r_1 \bowtie (r_2 \bowtie r_3)$$

(Join Associativity) $\bowtie$

If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose

$$(r_1 \bowtie r_2) \bowtie r_3$$

so that we compute and store a smaller temporary relation.

Join Ordering Example (Cont.)

Consider the expression

$$\Pi_{name,title}(\sigma_{dept\_name=\text{``Music''}}(instructor)\bowtie teaches)\bowtie\Pi_{course\_id,\ title}\ (course))))$$

Could compute

$teaches\ \bowtie\ \Pi_{course\_id,\ title}\ (course)$ first,
and join result with
$\sigma_{dept\_name=\text{``Music''}}\ (instructor)$

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 first compute

$\sigma_{dept\_name=\text{``Music''}}\ (instructor)\ \bowtie\ teaches$

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

Cost Estimation

- Cost of each operator (not detailed here)
  - 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

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 $r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$.

- 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: $S_1 \bowtie (S - S_1)$ 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

Join Order Optimization Algorithm

```
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 (if any) 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 + A.cost

    if cost < bestplan[S].cost
         bestplan[S].cost = cost
         bestplan[S].plan = plan;
   return bestplan[S]
```

Cost of Cost-based Optimization !

- With dynamic programming, the time complexity of join order optimization is $O(3^n)$.
    - With $n = 10$, this number is 59000 instead of 176 billion!
- Space complexity is $O(2^n)$
- System R restricts only to left-deep join trees - good for pipelined execution

Left Deep Join Trees

- In **left-deep join trees,** the right-hand-side input for each join is a relation, not the result of an intermediate join.
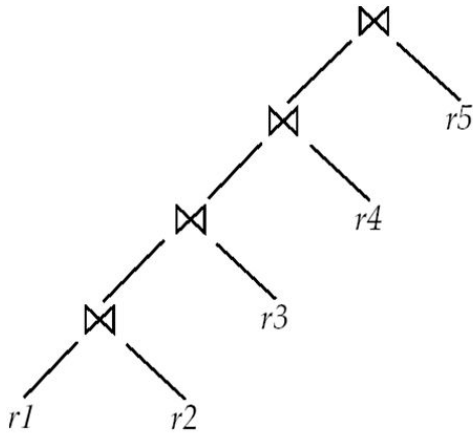
(a) Left-deep join tree
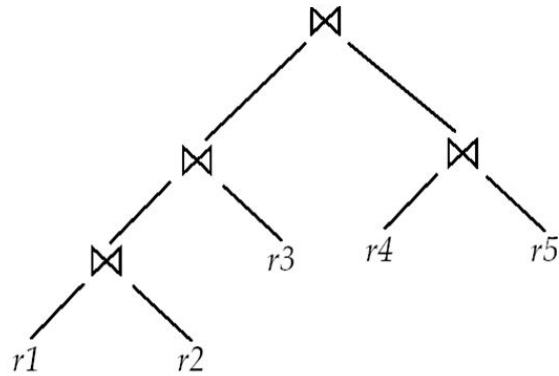
(b) Non-left-deep join tree

Left Deep Join Trees - Reduced Cost of Optimization

- 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.

- Time complexity of finding best join order is $O(n\, 2^n)$ - compared to (3^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)

Cost Based Optimization with Equivalence Rules

- **Physical equivalence rules:** extend equivalence rules with alternatives of algorithms.
- Efficient optimizer based on equivalent rules depends on
  - A space efficient representation of expressions which avoids making multiple copies of subexpressions
  - Efficient techniques for detecting duplicate derivations of expressions. It helps to use a minimal set of equivalence rules.
  - Dynamic programming, which **memorizes** the best plan for a subexpression the first time it is optimized, and reuses in on repeated optimization calls on same subexpression
  - Cost-based pruning techniques that avoid generating all plans

Volcano - recommended reading

The Volcano Optimizer Generator: Extensibility and Efficient Search

Goetz Graefe, Portland State University
William J. McKenna, University of Colorado at Boulder

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.2197&rep=rep1&type=pdf

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.

Structure of Query Optimizers (Cont.)

- Some query optimizers integrate heuristic selection and the generation of alternative access plans.
  - Frequently used approach
    - heuristic rewriting of nested block structure and aggregation
    - followed by cost-based join-order optimization for each block
- **Optimization cost budget** to stop optimization early (if cost of plan is less than cost of optimization)
- **Plan caching** to reuse previously computed plan if query is resubmitted
    - Even with different constants in query
- Even with the use of heuristics, cost-based query optimization imposes a substantial overhead.
  - But is worth it for expensive queries
  - Optimizers often use simple heuristics for very cheap queries, and perform exhaustive enumeration for more expensive queries

Credits

The slides of this lecture are taken from:

- Avi Silberschatz, Henry F. Korth, S. Sudarshan. Database System Concepts

ULB