Query Optimization

May 12, 2023

Overview

```sql
SELECT name, title
FROM instructor natural join teaches
     natural join course
WHERE dept_name ='Music';
```

1. Parse, check and verify the SQL
2. Translate into an RA query plan.
3. Query optimization: from an RA logical query plan to an optimized physical plan.

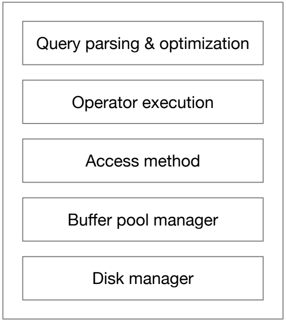
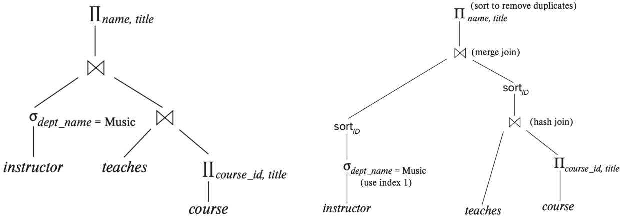


Figure: DBMS architecture

Agenda

```sql
SELECT name, title
FROM instructor natural join teaches
     natural join course
WHERE dept_name ='Music';
```



- Rule-based query rewriting: find better logical plans via RA equivalence rules.
- Cost-based query optimization: cost estimation and optimal join order search

Query optimizer

- Recall that SQL is declarative.
  - Users specify what tuples they want.
  - The query optimizer searches and picks the best query plan.
- Cost difference between query plans for a query can be huge.
- The first query optimizer was implemented in System R, in the 1970s.
- Many concepts and design decisions from the System R optimizer are still used today.

P. Selinger et al. (1979). Access Path Selection in a Relational Database Management System.

Rule-based Query Rewriting

RA equivalence rules (1)

- (i) $R \bowtie S = S \bowtie R$.   (ii) $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.
  - Natural join is commutative and associative (except for attributes ordering).
- $\sigma_\theta(R \times S) = R \bowtie_\theta S$. This rule converts a cross product to a theta join.
- $\Pi_{L_1}(\Pi_{L_2}(R)) = \Pi_{L_1}(R)$, where $L_1 \subseteq L_2$.
- $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_1 \wedge \theta_2}(R)$.

RA equivalence rules (2)

- Push down selection: $\sigma_{\theta_1 \wedge \theta_2}(R \bowtie_\theta S) = \sigma_{\theta_1}(R) \bowtie_\theta \sigma_{\theta_2}(S)$.

  Here θ_1 (resp. θ_2) involves only attributes of R (resp. S).

- Push down projection
  1. $\Pi_L(\sigma_\theta(R)) = \Pi_L(\sigma_\theta(\Pi_{L \cup L'}(R)))$
     - L' is the set of attributes that referenced by θ and not in L.
  2. $\Pi_L(R \bowtie_\theta S) = \Pi_L(\Pi_{L'}(R) \bowtie_\theta S)$.
     - L' consists of the set of attributes from R that either in L or referenced by θ.
  3. A symmetric version of (2).

Intuition: Have fewer tuples in a plan.

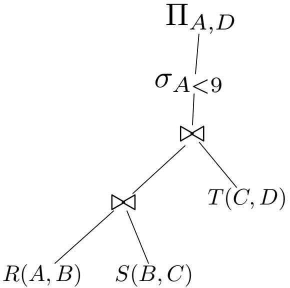
Rewrite logical plan via equivalence rules

SQL query

```sql
-- R(A,B), S(B,C), T(C,D)
SELECT R.A, S.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 9;
```

RA expression

$$\Pi_{A,D}(\sigma_{A<9}((R \bowtie S) \bowtie T))$$



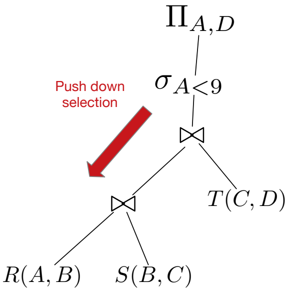
Rewrite logical plan via equivalence rules

SQL query

```sql
-- R(A,B), S(B,C), T(C,D)
SELECT R.A, S.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 9;
```

RA expression

$$\Pi_{A,D}(\sigma_{A<9}((R \bowtie S) \bowtie T))$$



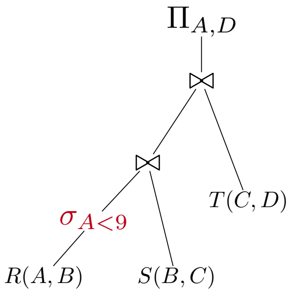
Rewrite logical plan via equivalence rules

SQL query

```
-- R(A,B), S(B,C), T(C,D)
SELECT R.A, S.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 9;
```

RA expression

$$\Pi_{A,D}((\sigma_{A<9}(R) \bowtie S) \bowtie T)$$



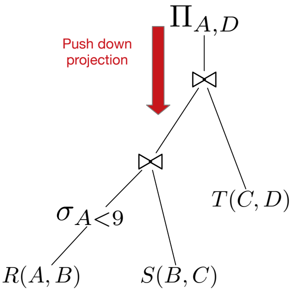
Rewrite logical plan via equivalence rules

SQL query

```
-- R(A,B), S(B,C), T(C,D)
SELECT R.A, S.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 9;
```

RA expression

$$\Pi_{A,D}((\sigma_{A<9}(R) \bowtie S) \bowtie T)$$



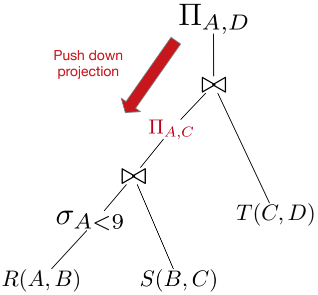
Rewrite logical plan via equivalence rules

SQL query

```
-- R(A,B), S(B,C), T(C,D)
SELECT R.A, S.D
FROM R,S,T
WHERE R.B = S.B
  AND S.C = T.C
  AND R.A < 9;
```

RA expression

$$\Pi_{A,D}(\Pi_{A,C}(\sigma_{A<9}(R) \bowtie S) \bowtie T)$$



Rules-based query optimization

1. Start with a logical plan.
2. Push selection/projection down as much as possible.
   - Pros: Reduce the size of intermediate results
   - Cons: Can be more expensive in some cases, e.g., joins filter better.
3. Join small relations first, and avoid cross products.
   - Pros: Reduce the size of intermediate results.
   - Cons: Size depends the join selectivity too.
4. Convert the transformed logical plan to a physical one
   – by choosing appropriate physical operators.

Cost-based Query Optimization

Cost estimation

- Plan cost = $\Sigma_{\text{Operator} \in \text{Plan}}$ (Operator cost)
- Operator cost $\propto$ Operator input size
- We have discussed how to estimate the cost of operators.
  - E.g., sequential/index scan, sort, joins.
- We still need to determine the size of operator input.
  - For base tables, equal to the size on disk.
  - For other operators, equal to "selectivity $\times$ size of children."

Statistics and catalog

- DBMS stores internal statistics about tables, attributes, and indexes in its internal catalog.

Notation	Statistics
$\|R\|$	number of tuples
$P(R)$	number of pages
$V(A, R)$	number of distinct values of A
$max(A, R)/min(A, R)$	max/min value of A
$H(A, R)$	Tree index height of A

Table: Selinger statistics for table R

- Catalogs are updated periodically.
- Modern DBMS use much more sophisticated stats.

Selection with equality predicates

$\sigma_{A=v}(R)$

- $|\sigma_{A=v}(R)| = |R|/V(A, R)$.
  - $|R|$: the number of tuples in R.
  - $V(A, R)$: the number of distinct values of A in R.
- Assumption: values of A are uniformly distributed in R.
- The selectivity factor of a predicate θ is the probability that a tuple in R satisfies θ.
- The selectivity factor of the predicate $A = v$ is $1/V(A, R)$.

Conjunctive predicates

$\sigma_{A=v \wedge B=u}(R)$

- $|\sigma_{A=v \wedge B=u}(R)| = |R|/V(A, R) * V(B, R)$
- The selectivity factor of $A = v \wedge B = u$ is $1/V(A, R) * V(B, R)$.
- Additional assumption:
  1. $A = v$ and $B = u$ are independent;
  2. No over-selection, i.e., both A and B are not keys.

Negative and disjunctive predicates

$\sigma_{A \neq v}(R)$

- Selectivity factor for $A \neq v$ is $1 - 1/V(A, R)$.
- Selectivity factor $\neg\theta$ is (1 - selectivity factor of θ).

$\sigma_{A=v \vee B=u}(R)$

- Selectivity factor: $1/V(A, R) + 1/V(B, R) - 1/V(A, R) * V(B, R)$
- Intuition: inclusion-exclusion principle.

Range predicates

$\sigma_{A<v}(R)$

- Suppose that $min(A, R)$ and $max(A, R)$ are available in catalog.
- If $v < min(R, A)$, the selectivity factor is 0
- Otherwise, the selectivity factor is $\frac{v-min(A,R)}{max(A-R)-min(A,R)}$
- $\sigma_{A \geqslant v}(R)$ can be estimated symmetrically.

Join size estimation

$R(A, B) \bowtie S(B, C)$

- Estimate the size of the product of $R \times S$ as $|R| * |S|$.
- Take $|R| * |S|/max(V(B, R), V(B, S))$ as the join size estimation.
- Assumption: containment of value sets.
  - If $V(B, R) < V(B, S)$, then $\Pi_B(R) \subseteq \Pi_B(S)$.
  - Not true in general. But holds in the common case of foreign key joins.
  - If $V(B, R) < V(B, S)$, then each tuple in R joins with $S/V(B, S)$ tuples of S.
  - Selectivity factor of $R.B = S.B$ is $1/max(V(B, R), V(B, S))$.
- Example. $|R| = 1000$, $|S| = 2000$, $\Pi(B, R) = 20$, $\Pi(B, S) = 50$.

  Then $|R \bowtie S| = 1000 * 5000/max(20, 50) = 40000$.

Estimation error

- Lots of assumptions and very rough estimation.
- Skewness is one of the main reasons that may lead to bad estimations.
- The assumption of mutual independence of the predicates may not hold!

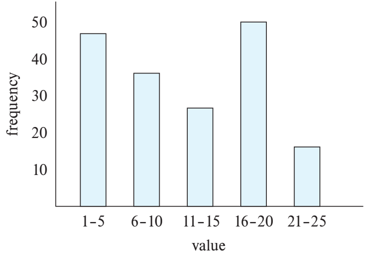
Example

Consider a table `employee(id, level, salary)`.

- Let `level` $\in [0, 10]$. Then selectivity of `level` > 6 is estimated as $\frac{10-6}{10-0} = 40\%$.
- Real selectivity is significantly lower than 40%, e.g., 20%.
- Assume that selectivity of `salary` > 400000 is 30%. Then what is the selectivity of `level` $> 6 \wedge$ `salary` > 400000 ?

Histograms

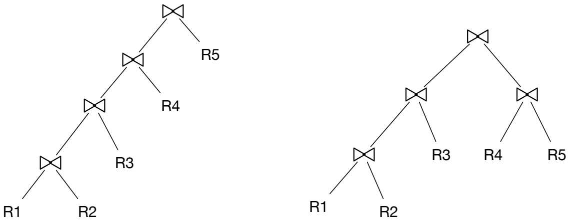
- Build histograms in the catalog to provide better estimation for common predicates over one or more columns.
- Equi-width: equal key ranges, store both key ranges and values.
- Equi-depth: histograms break up range such that each range has (approximately) the same number of tuples.
  - A equi-depth range example: (4, 8, 14, 19).



Cost-based plan search

- We have shown how to estimate the cost of one query plan.
- We next discuss how to pick the "best" one, i.e. the one with the lowest cost.
  - Enumerate all possible physical plans.
  - Pick the plan with the lowest cost.
- In practice, the goal is often not getting the optimal plan, but instead avoiding the really bad ones.
- We will focus on the search of optimal join orders.

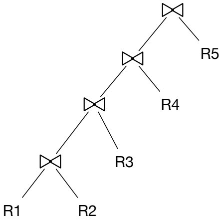
Join order



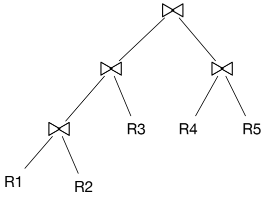
(a) Left-deep tree (b) Non-left-deep tree

- Recall that joins are commutative and associative.
- The search plan of join orders can be huge.
- In general, there $(2n-2)!/(n-1)!$ join orders for $R_1 \bowtie \cdots \bowtie R_n$.
  - With $n = 6$, the number is 30240.
  - With $n = 10$, the number is greater than 176 billion.

Reduce search space with left-deep joins



(a) Left-deep tree



(b) Non-left-deep tree

- In left-deep joins, only the left child can be a join operator.
- Left-deep joins allow to generate fully pipelined plans.
  - Intermediate results not written to temporary files.
  - Not all left-deep joins are fully pipelined, e.g., sort-merge join.
- There are $n!$ different leaf-deep join trees for $R_1 \bowtie \cdots R_n$.

  – $6! = 720$. Significantly fewer, but still lots.

Selinger algorithm

- First implemented in System R, frequently adapted and used.
- Use Selinger statistics for cost estimation.
- Only consider left-deep joins for plan enumeration.
- Generate optimal plans in a bottom-up fashion.

Patricia Selinger

P. Selinger et al. (1979). Access Path Selection in a Relational Database Management System.

Dynamic programming

We find the optimal left-deep join order of R_1, ..., R_n in a bottom-up fashion.

- Pass 1: Find the best single-table plan for R_1, ..., R_n.
- Pass 2: Find the best two-table plans for each pair of tables.
  This is done by combing best single table plans.
- ...
- Pass k: Find the best k-table plans for $\mathbb{S} \subseteq \{R_1, \ldots, R_n\}$ with $|\mathbb{S}| = k$.

$$\mathrm{Opt\_Cost}(\mathbb{S}) = \min_{R \in \mathbb{S}} \{\mathrm{Opt\_cost}(\mathbb{S} \setminus \{R\}) + \mathrm{Join\_cost}(\mathbb{S} \setminus \{R\}, R)\}$$

(i) Consider left-deep joins only. (ii) Pick the cheapest algorithm to join $(\mathbb{S} \setminus \{R\})$ and R.

Optimal substructure property. Any subplan of an optimal join plan must also be optimal.

Dynamic programming (cont'd)

Subset	Best Plan	Cost
$\{R\}$	IndexScan	100
$\{S\}$	SeqScan	80
$\{T\}$	IndexScan	50
$\{R, S\}$	HashJoin	160
$\{R, T\}$	MergeJoin	160
$\{S, T\}$	HashJoin	140
$\{R, S, T\}$	HashJoin	700

Table: DP table for $R \bowtie S \bowtie T$

Cost analysis: $n * 2^n$: (i) 2^n subsets in total; (ii) for each subset $\mathbb{S}$, we need to iterate through each element of each subset to find the optimal plan, which is at most n.

The need for interesting order

Example: $R(A, B) \bowtie S(A, C) \bowtie T(A, D)$

- Best plan for $R \bowtie S$: hash join (beats sort-merge join).
- Best overall plan for $R \bowtie S \bowtie T$ can be
  - First Sort-merge join R and S
  - Then sort-merge join $R \bowtie S$ with T.

  This can happen assuming that T is sorted on attribute A.
- Subplan of the optimal plan is not optimal.
- An intermediate result has an interesting order if it is sorted by anything that can be exploited by later processing.
  - The result of the sort-merge join of R and S is sorted on A.
  - This is an interesting order since a subsequent merge join of $R \bowtie S$ and T can utilize it.

Dealing with interesting orders

Subset	Best Plan	Interesting order	Cost
…	…	…	…
$\{R, S\}$	HashJoin	$\emptyset$	160
$\{R, S\}$	MergeJoin	$\{A\}$	200
…	…	…	…

Table: DP table for $R(A, B) \bowtie S(A, C) \bowtie T(A, D)$ with interesting order

- When picking the best plan
  - Comparing their cost is not enough
  - Comparing interesting orders is also needed
- Computes multiple optimal plans for each subset, one for each interesting order.
- Increases the complexity by factor $k + 1$, where k is the number of interesting orders.

Recap

- Rule-base query rewriting
  - Relational algebra equivalence rules
- Cost-based optimization
  - Need statistics to estimate sizes of intermediate results.
  - Dynamic programming for join orderings.

In practice, query optimization can be much more challenging.

Moerbotte and Neumann. Dynamic Programming Strikes Back. SIGMOD '08