

8 QUERY OPTIMIZATION

8.1 An Overview

- A query in a declarative language can be executed using a number of different ways that can generate the same result.
- **Objective:**
Find the “best” strategy for evaluating a given query.
- **Strategy** = an ordered sequence of operations (select, join, sort, ...)
- “Best” can be defined in several ways: minimal cost (resource usage), minimal response time, disk I/O, ...
- Extremely important - the performance of two strategies may differ by several orders of magnitude. Avoiding truly bad strategies is more important than finding the absolutely best.
- Two different approaches:
 - Heuristic query optimization
 - does not require/exploit statistical information about the underlying table
 - uses simple heuristic rules for finding a good strategy
 - simple but risky
 - Cost-based query optimization
 - tries to estimate the cost of a strategy
 - requires statistical information about the underlying tables
 - selects the strategy with the lowest estimated cost
- Once the best strategy has been found, it is converted to a detailed, self-contained **execution plan**.
The execution plan can be executed immediately and also stored away for later execution.

8.2 Equivalence Rules

Using the following equivalence rules, a number of equivalent strategies can be generated from a user query.

- Natural join operations are commutative:

$$r_1 \bowtie r_2 = r_2 \bowtie r_1 \quad (1)$$

- Natural join operations are associative:

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3) \quad (2)$$

- A join operation and a selection operation can be commuted:

$$\sigma_{p(r_1)}(r_1 \bowtie r_2) = (\sigma_{p(r_1)}r_1) \bowtie r_2 \quad (3)$$

where $p(r_1)$ is a predicate on r_1 .

- A projection operation and a join operation can be commuted:

$$\Pi_{A(r_1)}(r_1 \bowtie r_2) = (\Pi_{A(r_1)}r_1) \bowtie r_2 \quad (4)$$

where $A(r_1)$ is a subset of attributes in r_1 .

8.3 General Rules

- These rules do not depend on the statistical information about a database. They can be usually applied in all the situations.
- Basic idea:
Reduce the amount of data to be processed as quickly as possible.
- **Selection** reduces the number of tuples to be further processed
 - Perform selections as early as possible,
 - Perform selections before joins.

- **Projection** reduces the size of tuples, thereby reducing the amount of data to be carried to subsequent operations.
 - Perform projections as early as possible.
 - Retain only columns that
 - * appear in the result of the query or
 - * are needed in subsequent operations.

8.4 Optimization based on Cost Estimation

- By using the general rules, we may have a number of strategies generated for a user query.
- All the strategies are equivalent in the sense of generating the same result. However, they may have quite different costs.
- Optimization based on cost estimation involves estimating cost for each strategy and selecting the one with the lowest cost for execution.
- Cost estimation depends on database statistics.
- The total cost of a strategy consists of two parts

$$C = C_{io} + C_{cpu}$$

where C_{io} is the costs of disk I/O and C_{cpu} is the CPU cost.

- C_{io} is the sum of the costs for reading all the relation files from and writing the files to the disk.
- C_{cpu} is the costs for processing the data which have been in the main memory.
- When a strategy consists of a sequence of operations, C_{cpu} is the sum of the costs of all the operations.
- The CPU cost of an operation depends on
 - the nature of the operation,
 - the size of input, and

- the speed of the processor.
- When the input of an operation is the output of another operation, the size can be estimated by using the latter's **selectivity** and input size.
- More and more optimizers ignore the CPU costs.

8.5 Database Statistics

Database statistics are statistical information about the data stored in a database. For table r the statistics usually include

- n_r = number of tuples in relation r
- $\min(A)$ = the minimum value of attribute A
- $\max(A)$ = the maximum value of attribute A
- $\text{dist}(A)$ = number of distinct values of attribute A
- b_r = the number of blocks containing tuples of r
- s_r = the size of a tuple of r
- f_r = the number of tuples of r in a block.
- $\text{sc}(A,r)$ = the selection cardinality of attribute A of r .
- ht_r = the height of an index on r

8.6 Costs of selection

The I/O costs are machine-dependent. The following are the number of estimated disk reads. An I/O cost should be estimated as the product of the number and the cost of a disk read operation.

- **Linear search:** the number of reads is

$$b_r$$

- **Binary search:** the number of reads is

$$\text{ceiling}(\log_2(b_r)) + \text{ceiling}(sc(A, r)/f_r) - 1$$

- **Search with the primary index** and the search condition is an equality on key: the number of reads is

$$ht_r + 1$$

- **Search with the primary index** and the search condition is an equality on nonkey: the number of reads is

$$ht_r + \lceil sc(A, r)/f_r \rceil$$

where “ $\lceil \cdot \rceil$ ” represent “the ceiling of”.

- **Search with a secondary index** and the search condition is an equality: the number of reads is

$$ht_r + sc(A, r)$$

8.7 Costs of Join

- Nested-loop join:

– Algorithm

```

for each tuple tr in r do begin
  for each tuple ts in s do begin
    test pair (tr, ts) to see if they
      satisfy the join condition;
    if they do, add tr.ts to the result;
  end
end
end

```

– Cost:

The number of disk reads can be estimated as

$$b_r + n_r * b_s$$

- Indexed Nested-loop Join

- If an index is available on the inner loop's join attribute, index lookup can replace file scans to reduce the number of disk reads.
- This technique is mainly used for natural join or equal join.
- Cost:

$$b_r + n_r * C$$

where C is the cost of a single selection on relation s using an index.

8.8 Selectivity

- When the output of an operation is the input of some other operation, we need to estimate the output size (number of tuples, size of tuples)
<no of output tuples> = selectivity * <no of input tuples>
- Most database systems estimate selectivity under the (unrealistic) assumption of uniformity and independence.
- **Uniformity**: all values of an attribute occur with equal frequency.
- **Independence** the values of an attribute A are independent of the values of any other attribute B in the same table.
- For simplicity, assuming that attributes are of type INTEGER.

The following is selectivity of a selection operation with different predicates:

- $\text{sel}(A = \text{const}) = 1/\text{dist}(A)$
- $\text{sel}(A \leq \text{const}) = (\text{const} - \min(A)) / (\max(A) - \min(A))$
- $\text{sel}(A \geq \text{const}) = (\max(A) - \text{const}) / (\max(A) - \min(A))$

- Boolean expressions:

- $\text{sel}(P1 \text{ and } P2) = \text{sel}(P1) * \text{sel}(P2)$

- $\text{sel}(P1 \text{ or } P2) = \text{sel}(P1) + \text{sel}(P2)$
- $\text{sel}(\text{not } P) = 1 - \text{sel}(P)$
- Formulas are correct only if P1 and P2 are independent, i.e. no columns in common

8.9 Join Size Estimation

- Estimating the number of output tuples from a join.
- Virtually all joins are foreign key joins.

```

JoinSize( (r1.A1=r2.A2))
begin
  if A1 is a foreign key referencing r2.A2 then
    JoinSize = nr1;
  else
    if A2 is a foreign key referencing r1.A1 then
      JoinSize = nr2
    else
      prod = nr1 * nr2;
      JoinSize = min(prod/dist(A1), prod/dist(A2));
    endif
  endif
end

```

- If selection has been performed on the input to a join: first compute the joins size as if no selection had been performed then multiply the result with the estimated selectivity of the selection predicates of each input stream.