QUERY PROCESSING

Plans, Costs, and Optimization

University of Waterloo

List of Slides

Considerations

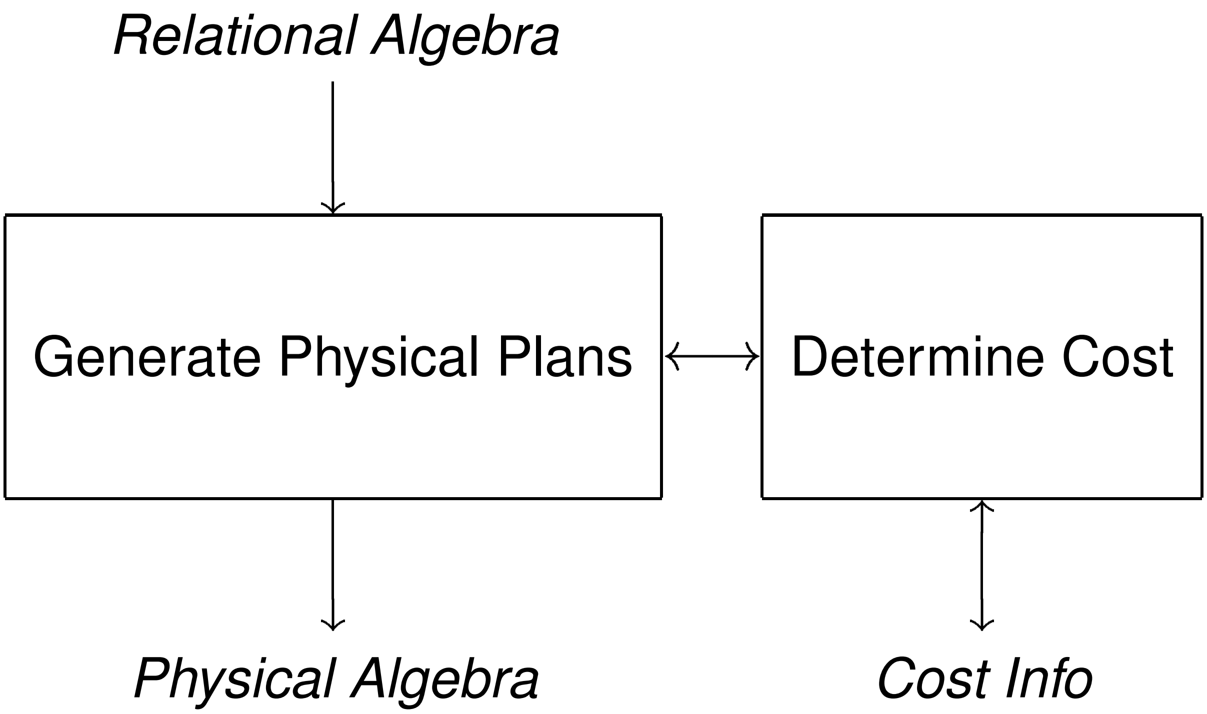
- Many possible **query plans** for a single query:
  1. equivalences in *Relational Algebra*
  2. choice of *Operator Implementation*

  $\Rightarrow$ performance differs greatly
- How do we choose the best plan?
  1. "always good" transformations
  2. cost-based model

  $\Rightarrow$ finding an **optimal plan** is computationally not feasible: we look for a *reasonable* one.

General Approach

- generate all physical plans equivalent to the query
- pick the one with the lowest cost



. . . All Equivalent Plans?!

- Cannot be done in general:

  $\Rightarrow$ it is **undecidable** if a query (un-)satisfiable equivalent to an *empty* plan.

- Very expensive even for **conjunctive** queries

  $\Rightarrow$ the *Join-ordering* problem

- In practice:

  $\Rightarrow$ only plans of certain form are considered (restrictions on the search space.)

  $\Rightarrow$ the goal is to eliminate the really bad ones.

. . . and Pick the Best one?!

- How do we determine which plan is the best one?

  ⇒ we cannot just run the plan to find out

  ⇒ instead we estimate the cost based on
  stats collected by the DBMS for all relations

- A **Simple Cost Model** for disk I/O; Assumptions:

  Uniformity: all possible values of an attribute are equally likely to appear in a relation.

  Independence: the likelihood that an attribute has a particular value (in a tuple) does not depend on values of other attributes.

A Simple Cost Model (cont.)

- For a stored relation R with an attribute A we keep:

  $|R|$: the cardinality of R (the number of tuples in R)

  $b(R)$: the blocking factor for R

  $\min(R, A)$: the minimum value for A in R

  $\max(R, A)$: the maximum value for A in R

  $\mathrm{distinct}(R, A)$: the number of distinct values of A

- Based on these values we try to estimate the **cost** of physical plans.

Cost of Retrieval

Mark(Studnum, Course, Assignnum, Mark)

```
SELECT Studnum, Mark
FROM   Mark
WHERE  Course = 'PHYS'
  AND Studnum = 100 AND Mark > 90
```

Indices:

- clustering index CourseInd on Course
- non-clustering index StudnumInd on Studnum

Assume:

- $|\text{Mark}| = 10000$
- $b(\text{Mark}) = 50$
- 500 different students
- 100 different courses
- 100 different marks

Strategy 1: Use CourseInd

Assuming *uniform distribution* of tuples over the courses, there will be about $|\mathrm{Mark}|$/100 = 100 tuples with **`Course = PHYS`**.

Searching the CourseInd index has a cost of 2. Retrieval of the 100 matching tuples adds a cost of $100/b(\mathrm{Mark})$ data blocks.

The total cost of 4.

Selection of N tuples from relation R using a clustered index has a cost of $2 + N/b(R)$.

Strategy 2: Use StudnumInd

Assuming *uniform distribution* of tuples over student numbers, there will be about $|\mathrm{Mark}|$/500 = 20 tuples for each student.

Searching the StudnumInd has a cost of 2. Since this is not a clustered index, we will make the pessimistic assumption that each matching record is on a separate data block, i.e., 20 blocks will need to be read.

The total cost is 22.

Selection of N tuples from relation R using a clustered index has a cost of $2 + N$.

Strategy 3: Scan the Relation

The relation occupies 10,000/50 = 200 blocks, so 200 block I/O operations will be required.

Selection of N tuples from relation R by scanning the entire relation has a cost of $|R|/b(R)$.

Cost of other Relational Operations

Costs of **physical** operations (in I/O's):

- Selection: $\mathrm{cost}(\sigma_c(E)) = (1 + \epsilon_c)\,\mathrm{cost}(E)$.
- Nested-Loop Join (R is the **outer** relation):

$$\mathrm{cost}(R \bowtie S) = \mathrm{cost}(R) + (|R|/b)\,\mathrm{cost}(S)$$

- Index Join (R is the outer relation, and S is the inner relation: B-tree with depth d_S):

$$\mathrm{cost}(R \bowtie S) = \mathrm{cost}(R) + d_S|R|$$

- Sort-Merge Join:

$$\mathrm{cost}(R \bowtie S) = \mathrm{cost}(\mathrm{sort}(R)) + \mathrm{cost}(\mathrm{sort}(S))$$

  where $\mathrm{cost}(\mathrm{sort}(E)) = \mathrm{cost}(E) + (|E|/b)\log(|E|/b)$.

- etc . . .

Size Estimation

In the cost estimation we need to know sizes of results of operations: we use the **selectivity**, defined, for a condition $\sigma_{\text{condition}}(R)$, as:

$$\text{sel}(\sigma_{\text{condition}}(R)) = \frac{|\sigma_{\text{condition}}(R)|}{|R|}$$

Again, the optimizer will *estimate* selectivity using simple rules based on its statistics:

$$\text{sel}(\sigma_{A=c}(R)) \approx \frac{1}{\text{distinct}(R, A)}$$

$$\text{sel}(\sigma_{A \leq c}(R)) \approx \frac{c - \min(R, A)}{\max(R, A) - \min(R, A)}$$

$$\text{sel}(\sigma_{A \geq c}(R)) \approx \frac{\max(R, A) - c}{\max(R, A) - \min(R, A)}$$

Size Estimation (cont.)

For Joins:

- General Join (on attribute A):

$$|R \bowtie S| \approx |R| \frac{|S|}{\text{distinct}(S, A)}$$

  or as

$$|R \bowtie S| \approx |S| \frac{|R|}{\text{distinct}(R, A)}$$

- Foreign key Join (Student and Enrolled joined on Sid):

$$|R \bowtie S| = |S| \frac{|R|}{|S|} = |R|$$

  May joins are foreign key joins, like this one.

More Advanced Statistics

- so far only a very primitive cost estimation approach
- in practice: more complex approaches
  - ⇒ histograms to approximate non-uniform distributions
  - ⇒ correlations between attributes
  - ⇒ uniqueness (keys) and containment (inclusions)
  - ⇒ sampling methods
  - ⇒ etc, etc

Plan Generation

- common approach:
  1. apply "always good" transformations

     ⇒ **heuristics** that work in the majority of cases
  2. cost-based join-order selection

     ⇒ applied on **conjunctive subqueries**

     the "select blocks"

     ⇒ still computationaly not tractable.

"Always good" transformations

- Push selections:

$$\sigma_\varphi(E_1 \bowtie_\theta E_2) = \sigma_\varphi(E_1) \bowtie_\theta E_2$$

  for φ involving columns of E_1 only (and vice versa).

- Push projections:

$$\pi_V(R \bowtie_\theta S) = \pi_V(\pi_{V_1}(R) \bowtie_\theta \pi_{V_2}(S))$$

  where V_1 is the set of all attributes of R involved in θ and V (similarly for V_2).

- Replace products by joins:

$$\sigma_\varphi(R \times S) = R \bowtie_\varphi S$$

$\Rightarrow$ also reduces the space of plans we need to search

Example

- Assume that
  - ⇒ there are $|S| = 1000$ students,
  - ⇒ enrolled in $|C| = 500$ classes.
  - ⇒ the enrollment table is $|E| = 5000$,
  - ⇒ and, on average, each student is registered for five courses.
- Then:

$$\mathrm{cost}(\sigma_{\mathrm{name}='\mathrm{Smith}'}(S \bowtie (E \bowtie C))) >> \mathrm{cost}(\sigma_{\mathrm{name}='\mathrm{Smith}'}(S) \bowtie (E \bowtie C))$$

Join Order Selection

- Joins are associative $R \bowtie S \bowtie T \bowtie U$ can be equivalently expressed as
  1. $((R \bowtie S) \bowtie T) \bowtie U$
  2. $(R \bowtie S) \bowtie (T \bowtie U)$
  3. $R \bowtie (S \bowtie (T \bowtie U))$

  $\Rightarrow$ try to minimize the intermediate result(s).
- Moreover, we need to decide which of the subexpressions is evaluated first

  $\Rightarrow$ e.g., Nested Loop join's cost is **not** symmetric!

Example

We have the following two join orders to pick from:

1. $\sigma_{\text{name}='\text{Smith}'}(S) \bowtie (E \bowtie C)$

   we produce $E \bowtie C$, which has one tuple for each course registration (by any student) $\sim$ 5000 tuples.

2. $(\sigma_{\text{name}='\text{Smith}'}(S) \bowtie E) \bowtie C$

   we produce an intermediate relation which has one tuple for each course registration by a student named Smith. If there are only a few Smith's among the 1,000 students (say there are 10), this relation will contain about 50 tuples.

Pipelined Plans

- all operators (except sorting) operate without storing intermediate results

  $\Rightarrow$ iterator protocols in constant storage
  $\Rightarrow$ no recomputation for **left-deep** plans

Temporary Store

- General pipelined plans lead to *recomputation*
- We introduce an additional **store** operator

  ⇒ allows us to store intermediate results in a relation
  ⇒ we can also built a (hash) index on top of the result
- Semantically, the operator represents the **identity**
- The costs of plans:
  1. cumulative cost—to compute the value of the expression and store then in a relation (once):

     $$\mathrm{cost}_c(\textsf{store}(E)) = \mathrm{cost}_c(E) + \mathrm{cost}_s(E) + |E|/b$$

  2. scanning cost—to “read” all the tuples in the stored result of the expression:

     $$\mathrm{cost}_s(\textsf{store}(E)) = |E|/b$$

Summary

- Query plans represented in the relational algebra may be transformed using simple transformation rules.

  $\Rightarrow$ but there are too many of those
  we need to limit the search space (e.g., pruning)

- Using collected statistics about base relations, the details (e.g., selection method) of a plan may be determined, and its cost may be estimated.

- The optimizer selects a low-cost plan for execution.