QUERY PROCESSING

Plans, Costs, and Optimization

University of Waterloo
List of Slides

1
2 Considerations
3 General Approach
4 . . . All Equivalent Plans?!
5 . . . and Pick the Best one?!
6 A Simple Cost Model (cont.)
7 Cost of Retrieval
8 Strategy 1: Use CourseInd
9 Strategy 2: Use StudnumInd
10 Strategy 3: Scan the Relation
11 Cost of other Relational Operations
12 Size Estimation
13 Size Estimation (cont.)
14 More Advanced Statistics
15 Plan Generation
16 “Always good” transformations
17 Example
18 Join Order Selection
19 Example
20 Pipelined Plans
21 Temporary Store
22 Summary
Considerations

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

- How do we choose the best plan?
  1. “always good” transformations
  2. cost-based model
     ⇒ 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

Relational Algebra

Generate Physical Plans

Physical Algebra

Cost Info

Determine Cost
All Equivalent Plans?!

- Cannot be done in general:
  - it is **undecidable** if a query (un-)satisfiable equivalent to an *empty* plan.

- Very expensive even for **conjunctive** queries
  - the *Join-ordering* problem

- In practice:
  - only plans of certain form are considered (restrictions on the search space.)
  - 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 $R$
  - $\text{distinct}(R, A)$: 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 $|\text{Mark}|/100 = 100$ tuples with Course = PHYS.

Searching the CourseInd index has a cost of 2. Retrieval of the 100 matching tuples adds a cost of $100/b(\text{Mark})$ 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 $\frac{|\text{Mark}|}{500} = 20$ 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:** \( \text{cost}(\sigma_c(E)) = (1 + \epsilon_c) \text{cost}(E) \).  
- **Nested-Loop Join** (\( R \) is the **outer** relation):
  \[
  \text{cost}(R \Join S) = \text{cost}(R) + (|R|/b) \text{cost}(S)
  \]
- **Index Join** (\( R \) is the outer relation, and \( S \) is the inner relation: B-tree with depth \( d_S \)):
  \[
  \text{cost}(R \Join S) = \text{cost}(R) + d_S |R|
  \]
- **Sort-Merge Join**:
  \[
  \text{cost}(R \Join S) = \text{cost}(\text{sort}(R)) + \text{cost}(\text{sort}(S))
  \]
  where \( \text{cost}(\text{sort}(E)) = \text{cost}(E) + (|E|/b) \log(|E|/b) \).
- 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 $A$):

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

  or as

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

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

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

  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 computationally not tractable.
“Always good” transformations

- Push selections:

\[ \sigma_\varphi(E_1 \land_\theta E_2) = \sigma_\varphi(E_1) \land_\theta E_2 \]

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

- Push projections:

\[ \pi_V(R \land_\theta S) = \pi_V(\pi_{V_1}(R) \land_\theta \pi_{V_2}(S)) \]

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

- Replace products by joins:

\[ \sigma_\varphi(R \times S) = R \land_\varphi S \]

\( \Rightarrow \) 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:

\[
\text{cost}(\sigma_{\text{name}='\text{Smith}'}(S \Join (E \Join C))) >> \\
\text{cost}(\sigma_{\text{name}='\text{Smith}'}(S) \Join (E \Join C))
\]
Join Order Selection

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

  ⇒ try to minimize the intermediate result(s).

- Moreover, we need to decide which of the subexpressions is evaluated first

  ⇒ 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
  ⇒ iterator protocols in constant storage
  ⇒ no recomputation for left-deep plans
Temporary Store

- General pipelined plans lead to \textit{recomputation}
- We introduce an additional \texttt{store} operator
  \(\Rightarrow\) allows us to store intermediate results in a relation
  \(\Rightarrow\) we can also build a (hash) index on top of the result
- Semantically, the operator represents the \texttt{identity}
- The costs of plans:
  1. cumulative cost—to compute the value of the expression and store then in a relation (once):
     \[
     \text{cost}_c(\text{store}(E)) = \text{cost}_c(E) + \text{cost}_s(E) + \frac{|E|}{b}
     \]
  2. scanning cost—to “read” all the tuples in the stored result of the expression:
     \[
     \text{cost}_s(\text{store}(E)) = \frac{|E|}{b}
     \]
Summary

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