Why Query Optimization?

Queries must be executed and execution takes time

There are multiple execution plans for most queries

Some plans cost less than others

How to come up with the right query plan?

Measure the cost of each query
Enumerate possibilities
Pick the least expensive one

Is that all?

Simple Example

```
SELECT * FROM A, B, C WHERE A.n = B.n AND B.m = C.m
A = 100 tuples
B = 50 tuples
C = 2 tuples
Which plan is cheaper?
Join( C, Join( A, B ) )
Join( A, Join( B, C ) )
```

But the search space is too big

Search space becomes too large as the number of joins increases
In a matter of n!

Just to remind you,
20! = 2,432,902,008,176,640,000

So now what do we do?

Discussion 1

• What will be changed in the query optimizer with the hardware today?
1. Can the search on the query space be decomposed into sub-problems, i.e., could we design a parallel program for the space search problem?
2. More broadly, is there some way that we take advantage of the new multi-core technology to improve query optimization?
Changes Brought by Multi-core to Query Optimizer

- [VLDB 2008] Parallelizing Query Optimization
- [SIGMOD 2009] Parallelizing Extensible Query Optimizations
- [SIGMOD 2009] Dependency-Aware Reordering for Parallelizing Query Optimization in Multi-Core CPUs

Use Statistics

Selectivity factor
- Expected fraction of tuples that satisfy the predicate

For each relation keep track of
- Cardinality of relations
- Number of pages in the segment containing tuples of relation T
- Fraction of empty to non-empty pages

Use these statistics in conjunction with
- Sargable predicates
- Interesting Orders

But...

- Statistics alone cannot save us
  - Expensive to compute
  - Can’t keep track of all joint statistics

- Compromise on statistics
  - Periodically update stats for each relation

Predicates

- Predicates like =, >, NOT, etc. reduce the number of tuples

- THUS: Evaluate predicates as early as possible

System R Optimizer

- A join operator can use either
  - nested loop implementation
  - sort-merge implementation

- Predicates are evaluated as early as possible

System R Optimizer

- Cost model relies on
  - Use of statistics
  - Selectivity factor
  - CPU and I/O cost

- Consideration of interesting orders
Interesting Orders

GROUP BY and ORDER BY or sort-merge joins generate interesting orders

Cost of generating the interesting order should be added to the cost of a plan

Back to the issue of search space

Recall: search space too big (n!)

Dynamic programming approach

Dynamic programming (Wikipedia)

Assumption: Cost model satisfies the principle of optimality

*Optimal substructure* means that optimal solutions of subproblems can be used to find the optimal solutions of the overall problem.

Dynamic programming

An N-Join is really just a sequence of 2-Joins

Break the problem into smaller sub-problems. Solve these problems optimally using this three-step process recursively. Use these optimal solutions to construct an optimal solution for the original problem.

Discussion 2

- With formulas & methods;
- Without the estimation & experiment

1. Similar to Codd's paper "A Relational Model of Data for Large Shared Data Banks", this paper was also published in the ACM even though it lacks experimental data. Can we say that Codd's paper perhaps caused a paradigm shift in how the journal accepts papers?

2. It is the first time that a cost-based method is proposed to select access paths. The method has a deep influence to the subsequent work on query optimizer. But at that time when the paper is published, do you think the method proposed in the paper is not convincing without experimental data? If so, why does it succeed in the following decades?

Major Contributions of Paper

- Cost based optimization
- Statistics
- CPU utilization (for sorts, etc.)
- Dynamic programming approach
- Interesting Orders
Summary of the Approach

Enumerate access paths to each relation
  - Sequential scans
  - Interesting orders
Enumerate access paths to join a second relation to these results (if there is a predicate to do so)
  - Nested loop (unordered)
  - Merge (interesting order)
Compare with equivalent solutions found so far but only keep the cheapest

Example Query

```
SELECT NAME, TITLE, SAL, DNAME
FROM EMP, DEPT, JOB
WHERE TITLE='CLERK'
AND LOC='DENVER'
AND EMP.DNO=DEPT.DNO
AND EMP.JOB=JOB.JOB
```

Example Schema

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>DNO</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWIT</td>
<td>10</td>
<td>12</td>
<td>7600</td>
</tr>
<tr>
<td>JONES</td>
<td>10</td>
<td>6</td>
<td>16000</td>
</tr>
<tr>
<td>DOE</td>
<td>10</td>
<td>9</td>
<td>8000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Example Initial Access Paths for single relations

Example Search Tree

2nd Relations Nested Loop