Lecture #7

Query Optimization
May 16th, 2002

Agenda/Administration

- Exam date set: June 10th, 6:30pm. Place TBA.
- Volunteers for presenting projects during last class.
- Project demos. Schedules coming soon.

Query Optimization

- Major issues:
  - Transformations (we saw a few, more coming)
  - Un-nesting of subqueries; magic-set transformations.
  - Join ordering
  - Maintaining statistics
  - General architectural issues to deal with large search space.

Schema for Some Examples

- Sailors (sid: integer, sname: string, rating: integer, age: real)
- Reserves (sid: integer, bid: integer, day: dates, rname: string)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages (4000 tuples)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages (4000 tuples).

Rewrites: Group By and Join

- Schema:
  - Product (pid, unitprice,…)
  - Sales(tid, date, store, pid, units)

- Trees:
  - GroupBy(pid)
  - Sum(units)
  - Join
    - Products
    - Filter (in NW)
    - Filter (date in Q2, 2000)

Rewrites: Operation Introduction

- Schema: (pid determines cid)
  - Category (pid, cid, details)
  - Sales(tid, date, store, pid, amount)

- Trees:
  - GroupBy(cid)
  - Sum(amount)
  - Join
    - Category
    - Filter (…)
    - Scansales
    - Filter (store in {CA, WA})
Query Rewriting: Predicate Pushdown

The earlier we process selections, less tuples we need to manipulate higher up in the tree.

Disadvantages?

Query Rewrites: Predicate Pushdown (through grouping)

Select bid, Max(age)
From Reserves R, Sailors S
Where R.sid=S.sid
GroupBy bid
Having Max(age) > 40

- For each boat, find the maximal age of sailors who’ve reserved it.
- Advantage: the size of the join will be smaller.
- Requires transformation rules specific to the grouping/aggregation operators.
- Will it work work if we replace Max by Min?

Query Rewrite: Predicate Movearound

Sailing wiz dates: when did the youngest of each sailor level rent boats?

First, move predicates up the tree.

Select sid, date
From V1, V2
Where V1.rating = V2.rating and V1.age = V2.age

Create View V1 AS
Select rating, Min(age)
From Sailors S
Where S.age < 20
Group By rating

Create View V2 AS
Select sid, rating, age, date
From Sailors S, Reserves R
Where R.sid=S.sid

Select sid, date
From V1, V2
Where V1.rating = V2.rating and V1.age = V2.age, age < 20

Query Rewrite Summary

- The optimizer can use any semantically correct rule to transform one query to another.
- Rules try to:
  - move constraints between blocks (because each will be optimized separately)
  - Unnest blocks
- Especially important in decision support applications where queries are very complex.
- In a few minutes of thought, you’ll come up with your own rewrite. Some query, somewhere, will benefit from it.
- Theorems?
Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
    - Depends on input cardinalities.
  - Must estimate size of result for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.
- We’ll discuss the System R cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

Statistics and Catalogs

- Need information about the relations and indexes involved. Catalogs typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Cost Model for Our Analysis

- As a good approximation, we ignore CPU costs:
  - B: The number of data pages
  - P: Number of tuples per page
  - D: (Average) time to read or write disk page
- Measuring number of page I/O’s ignores gains of pre-fetching blocks of pages; thus, even I/O cost is only approximated.

Size Estimation and Reduction Factors

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- Reduction factor (RF) associated with each term reflects the impact of the term in reducing result size. Result cardinality = Max # tuples * product of all RF’s.
  - Implicit assumption that terms are independent!
  - Term col=value has RF 1/NKeys(I), given index I on col
  - Term col=col2 has RF 1/MAX(NKeys(I1), NKeys(I2))
  - Term col>value has RF (High(I)-value)/(High(I)-Low(I))

Histograms

- Key to obtaining good cost and size estimates.
- Come in several flavors:
  - Equi-depth
  - Equi-width
- Which is better?
- Compressed histograms: special treatment of frequent values.
**Histograms**

Employee(ssn, name, salary, phone)

- Maintain a histogram on salary:

<table>
<thead>
<tr>
<th>Salary</th>
<th>Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..2k</td>
<td>200</td>
</tr>
<tr>
<td>2k..4k</td>
<td>800</td>
</tr>
<tr>
<td>4k..6k</td>
<td>5000</td>
</tr>
<tr>
<td>6k..8k</td>
<td>12000</td>
</tr>
<tr>
<td>8k..10k</td>
<td>6500</td>
</tr>
<tr>
<td>&gt;10k</td>
<td>500</td>
</tr>
</tbody>
</table>

- \(T(Employee) = 25000\), but now we know the distribution

**Histograms**

Ranks(rankName, salary)

- Estimate the size of Employee \(\gg_{salary}\) Ranks

<table>
<thead>
<tr>
<th>Salary</th>
<th>Employee</th>
<th>Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..2k</td>
<td>200</td>
<td>500</td>
</tr>
<tr>
<td>2k..4k</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>4k..6k</td>
<td>5000</td>
<td>5000</td>
</tr>
<tr>
<td>6k..8k</td>
<td>12000</td>
<td>12000</td>
</tr>
<tr>
<td>8k..10k</td>
<td>6500</td>
<td>4000</td>
</tr>
<tr>
<td>&gt;10k</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

**Plans for Single-Relation Queries**

(Prep for Join ordering)

- **Task**: create a query execution plan for a single Select-project-group-by block.
- **Key idea**: consider each possible access path to the relevant tuples of the relation. Choose the cheapest one.
- The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are pipelined into the aggregate computation).

**Example**

```sql
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```

**Determining Join Ordering**

- \(R_1 \bowtie R_2 \bowtie ... \bowtie R_n\)
- Join tree:

```
R3  
R1  
R2  
R4  
```
- A join tree represents a plan. An optimizer needs to inspect many (all?) join trees
Types of Join Trees

- Left deep:

```
   R1 -> R2 -> R3 -> R4
      |     |     |
     R5   R2   R4
```

- Bushy:

```
   R1 -> R2
   R3
```

- Right deep:

```
   R1 -> R2 -> R3
      |     |     |
     R5   R2   R4
```

Problem

- Given: a query $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- Assume we have a function $\text{cost()}$ that gives us the cost of every join tree
- Find the best join tree for the query
- How?

Dynamic Programming

- Idea: for each subset of $\{R_1, \ldots, R_n\}$, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for $\{R_1\}$, $\{R_2\}$, ..., $\{R_n\}$
  - Step 2: for $\{R_1, R_2\}$, $\{R_1, R_3\}$, ..., $\{R_{n-1}, R_n\}$
  - ...
  - Step n: for $\{R_1, \ldots, R_n\}$
- A subset of $\{R_1, \ldots, R_n\}$ is also called a subquery

For each subquery $Q \subseteq \{R_1, \ldots, R_n\}$ compute the following:

- $\text{Size}(Q)$
- A best plan for $Q$: $\text{Plan}(Q)$
- The cost of that plan: $\text{Cost}(Q)$
Dynamic Programming

- **Step 1**: For each \{R_i\} do:
  - \text{Size}(\{R_i\}) = B(R_i)
  - \text{Plan}(\{R_i\}) = R_i
  - \text{Cost}(\{R_i\}) = \text{(cost of scanning } R_i)\)

- **Step i**: For each \(Q \subseteq \{R_1, \ldots, R_n\}\) of cardinality \(i\) do:
  - Compute \text{Size}(Q) (later…)
  - For every pair of subqueries \(Q', Q''\) s.t. \(Q = Q' \cup Q''\)
    - \text{compute cost(Plan}(Q') \bowtie \text{Plan}(Q''))
  - \text{Cost}(Q) = \text{the smallest such cost}
  - \text{Plan}(Q) = \text{the corresponding plan}

- Return Plan(\{R_1, \ldots, R_n\})

Summary: computes optimal plans for subqueries:
- Step 1: \{R_1\}, \{R_2\}, \ldots, \{R_n\}
- Step 2: \{R_1, R_2\}, \{R_1, R_3\}, \ldots, \{R_{n-1}, R_n\}
- ...
- Step n: \{R_1, \ldots, R_n\}

- We used naïve size/cost estimations
- In practice:
  - more realistic size/cost estimations (next)
  - heuristics for Reducing the Search Space
    - Restrict to left linear trees
    - Restrict to trees “without cartesian product”
    - need more than just one plan for each subquery
    - “interesting orders”
- Why did it work?

Query Optimization

- We’re done.
- Questions? Comments?

What is Data Integration

- **Providing**
  - Uniform (same query interface to all sources)
  - Access to (queries eventually updates too)
  - Multiple (we want many, but 2 is hard too)
  - Autonomous (DBA doesn’t report to you)
  - Heterogeneous (data models are different)
  - Structured (or at least semi-structured)
  - Data Sources (not only databases).
The Problem: Data Integration

Uniform query capability across autonomous, heterogeneous data sources on LAN, WAN, or Internet

Motivation(s)

- **Enterprise** data integration; web-site construction.
- **WWW:**
  - Comparison shopping
  - Portals integrating data from multiple sources
  - B2B, electronic marketplaces
- **Science and culture:**
  - Medical genetics: integrating genomic data
  - Astrophysics: monitoring events in the sky.
  - Environment: Puget Sound Regional Synthesis Model
  - Culture: uniform access to all cultural databases produced by countries in Europe.

And if that wasn’t enough…

- Explosion of intranet and extranet information
- 80% of corporate information is unmanaged
- By 2004 30X more enterprise data than 1999
- The average company:
  - maintains 49 distinct enterprise applications
  - spends 35% of total IT budget on integration-related efforts

Discussion

- Why is it hard?
- How will we solve it?

Current Solutions

- **Mostly ad-hoc programming:** create a special solution for every case; pay consultants a lot of money.
- **Data warehousing:** load all the data periodically into a warehouse.
  - 6-18 months lead time
  - Separates *operational* DBMS from *decision support* DBMS. (not only a solution to data integration).
  - Performance is good; data may not be fresh.
  - Need to clean, scrub you data.

Data Warehouse Architecture

- User queries
  - OLAP / Decision support / Data cubes / data mining

- Relational database (warehouse)

- Data source
  - Data extraction programs

- Data cleaning / scrubbing

- Data source
The Virtual Integration Architecture

- Leave the data in the sources.
- When a query comes in:
  - Determine the relevant sources to the query
  - Break down the query into sub-queries for the sources.
  - Get the answers from the sources, and combine them appropriately.
- Data is fresh.
- Challenge: performance.

Virtual Integration Architecture

<table>
<thead>
<tr>
<th>wrap</th>
<th>Data source</th>
<th>wrap</th>
<th>Data source</th>
<th>wrap</th>
<th>Data source</th>
</tr>
</thead>
</table>

Sources can be: relational, hierarchial (IMS), structure files, web sites.

Research Projects

- Garlic (IBM)
- Information Manifold (AT&T)
- Tsimmis, InfoMaster (Stanford)
- The Internet Softbot/Razor/Tukwila (UW)
- Hermes (Maryland)
- DISCO (INRIA, France)
- SIMS/Ariadne (USC/ISI)

Dimensions to Consider

- How many sources are we accessing?
- How autonomous are they?
- Meta-data about sources?
- Is the data structured?
- Queries or also updates?
- Requirements: accuracy, completeness, performance, handling inconsistencies.
- Closed world assumption vs. open world?

Outline

- Wrappers
- Semantic integration and source descriptions:
  - Modeling source completeness
  - Modeling source capabilities
- Query optimization
- Query execution (mostly Zack)

Wrapper Programs

- Task: to communicate with the data sources and do format translations.
- They are built w.r.t. a specific source.
- They can sit either at the source or at the mediator.
- Often hard to build (very little science).
- Can be “intelligent”: perform source-specific optimizations.
Example

Transform:

```xml
<book>
  <title>Introduction to DB</title>
  <author>Phil Bernstein</author>
  <author>Eric Newcomer</author>
  <publisher>Addison Wesley</publisher>
  <year>1999</year>
</book>
```

into:

```xml
<book>
  <title>Introduction to DB</title>
  <author>Phil Bernstein</author>
  <author>Eric Newcomer</author>
  <publisher>Addison Wesley</publisher>
  <year>1999</year>
</book>
```

Data Source Catalog

- Contains all meta-information about the sources:
  - Logical source contents (books, new cars).
  - Source capabilities (can answer SQL queries)
  - Source completeness (has all books).
  - Physical properties of source and network.
  - Statistics about the data (like in an RDBMS)
  - Source reliability
  - Mirror sources
  - Update frequency.

Content Descriptions

- User queries refer to the *mediated schema*.
- Data is stored in the sources in a *local schema*.
- Content descriptions provide the semantic mappings between the different schemas.
- Data integration system uses the descriptions to translate user queries into queries on the sources.

Desiderata from Source Descriptions

- **Expressive power**: distinguish between sources with closely related data. Hence, be able to prune access to irrelevant sources.
- **Easy addition**: make it easy to add new data sources.
- **Reformulation**: be able to reformulate a user query into a query on the sources efficiently and effectively.

Reformulation Problem

- **Given**:
  - A query Q posed over the mediated schema
  - Descriptions of the data sources
- **Find**:
  - A query Q’ over the data source relations, such that:
    - Q’ provides only *correct answers* to Q, and
    - Q’ provides all possible answers from to Q given the sources.

Approaches to Specifying Source Descriptions

- **Global-as-view**: express the mediated schema relations as a set of views over the data source relations
- **Local-as-view**: express the source relations as views over the mediated schema.
- Can be combined with no additional cost.
Global-as-View

Mediated schema:
- Movie(title, dir, year, genre),
- Schedule(cinema, title, time).

Create View Movie AS

\[
\begin{align*}
&\text{select}^* \text{ from } S_1 \quad [S_1(title, dir, year, genre)] \\
&\text{union} \\
&\text{select}^* \text{ from } S_2 \quad [S_2(title, dir, year, genre)] \\
&\text{union} \\
&\text{select } S_3.title, S_3.dir, S_4.year, S_4.genre \\
&\text{from } S_3, S_4 \\
&\text{where } S_3.title=S_4.title
\end{align*}
\]

Global-as-View: Example 2

Mediated schema:
- Movie(title, dir, year, genre),
- Schedule(cinema, title, time).

Create View Movie AS $[S_1(title, dir, year)]$

\[
\begin{align*}
&\text{select } \text{title, dir, year, NULL} \\
&\text{from } S_1 \\
&\text{union} \\
&\text{select } \text{title, dir, NULL, genre} \\
&\text{from } S_2
\end{align*}
\]

Global-as-View: Example 3

Mediated schema:
- Movie(title, dir, year, genre),
- Schedule(cinema, title, time).

Source S4: S4(cinema, genre)

Create View Movie AS

\[
\begin{align*}
&\text{select NULL, NULL, NULL, genre} \\
&\text{from } S_4
\end{align*}
\]

Create View Schedule AS

\[
\begin{align*}
&\text{select } \text{cinema, NULL, NULL} \\
&\text{from } S_4
\end{align*}
\]

Global-as-View Summary

• Query reformulation boils down to view unfolding.
• Very easy conceptually.
• Can build hierarchies of mediated schemas.
• You sometimes lose information. Not always natural.
• Adding sources is hard. Need to consider all other sources that are available.

Local-as-View: example 1

Mediated schema:
- Movie(title, dir, year, genre),
- Schedule(cinema, title, time).

Create Source S1 AS

\[
\begin{align*}
&\text{select}^* \text{ from Movie}
\end{align*}
\]

Create Source S3 AS $[S_3(title, dir)]$

\[
\begin{align*}
&\text{select } \text{title, dir} \text{ from Movie}
\end{align*}
\]

Create Source S5 AS

\[
\begin{align*}
&\text{select } \text{title, dir, year} \\
&\text{from Movie} \\
&\text{where } \text{year} > 1960 \text{ AND genre} = \text{“Comedy”}
\end{align*}
\]

Local-as-View: Example 2

Mediated schema:
- Movie(title, dir, year, genre),
- Schedule(cinema, title, time).

Source S4: S4(cinema, genre)

Create Source S4

\[
\begin{align*}
&\text{select } \text{cinema, genre} \\
&\text{from Movie } m, \text{ Schedule } s \\
&\text{where } m.\text{title} = s.\text{title}
\end{align*}
\]

Now if we want to find which cinemas are playing comedies, there is hope!
Local-as-View Summary

- Very flexible. You have the power of the entire query language to define the contents of the source.
- Hence, can easily distinguish between contents of closely related sources.
- Adding sources is easy: they're independent of each other.
- Query reformulation: answering queries using views!

The General Problem

- Given a set of views V1,…,Vn, and a query Q, can we answer Q using only the answers to V1….Vn?
  - Many, many papers on this problem.
  - Great survey on the topic: (Halevy, 2000).

Local Completeness Information

- If sources are incomplete, we need to look at each one of them.
- Often, sources are locally complete.
- Movie(title, director, year) complete for years after 1960, or for American directors.
- Question: given a set of local completeness statements, is a query Q’ a complete answer to Q?

Example

- Movie(title, director, year) (complete after 1960).
- Show(title, theater, city, hour)
- Query: find movies (and directors) playing in Seattle:
  
  Select m.title, m.director
  From Movie m, Show s
  Where m.title=s.title AND city="Seattle"

- Complete or not?

Example #2

- Movie(title, director, year), Oscar(title, year)
- Query: find directors whose movies won Oscars after 1965:
  
  select m.director
  from Movie m, Oscar o
  where m.title=o.title AND m.year=o.year
  AND o.year > 1965.
- Complete or not?

Query Optimization

- Very related to query reformulation!
- Goal of the optimizer: find a physical plan with minimal cost.
- Key components in optimization:
  - Search space of plans
  - Search strategy
  - Cost model
Optimization in Distributed DBMS

- A distributed database (2-minute tutorial):
  - Data is distributed over multiple nodes, but is uniform.
  - Query execution can be distributed to sites.
  - Communication costs are significant.
- Consequences for optimization:
  - Optimizer needs to decide locality
  - Need to exploit independent parallelism.
  - Need operators that reduce communication costs (semi-joins).

DDBMS vs. Data Integration

- In a DDBMS, data is distributed over a set of uniform sites with precise rules.
- In a data integration context:
  - Data sources may provide only limited access patterns to the data.
  - Data sources may have additional query capabilities.
  - Cost of answering queries at sources unknown.
  - Statistics about data unknown.
  - Transfer rates unpredictable.

Modeling Source Capabilities

- Negative capabilities:
  - A web site may require certain inputs (in an HTML form).
  - Need to consider only valid query execution plans.
- Positive capabilities:
  - A source may be an ODBC compliant system.
  - Need to decide placement of operations according to capabilities.
- Problem: how to describe and exploit source capabilities.

Example #1: Access Patterns

Mediated schema relation: Cites(paper1, paper2)

Create Source S1 as
select *
from Cites
given paper1
Create Source S2 as
select paper1
from Cites
Select p1
From S1, S2
Where S2.paper1=S1.paper1 AND S1.paper2="Hal00"

Example #2: Access Patterns

Create Source S1 as
select *
from Cites
given paper1
Create Source S2 as
select paperID
from UW-Papers
Create Source S3 as
select paperID
from AwardPapers
given paperID
Query: select * from AwardPapers
Example #2: Solutions
- Can’t go directly to S3 because it requires a binding.
- Can go to S1, get UW papers, and check if they’re in S3.
- Can go to S1, get UW papers, feed them into S2, and feed the results into S3.
- Can go to S1, feed results into S2, feed results into S2 again, and then feed results into S3.
- Strictly speaking, we can’t a priori decide when to stop.
- Need recursive query processing.

Handling Positive Capabilities
- Characterizing positive capabilities:
  – Schema independent (e.g., can always perform joins, selections).
  – Schema dependent: can join R and S, but not T.
  – Given a query, tells you whether it can be handled.
- Key issue: how do you search for plans?
- Garlic approach (IBM): Given a query, STAR rules determine which subqueries are executable by the sources. Then proceed bottom-up as in System-R.

Matching Object Across Sources
- How do I know that A. Halevy in source 1 is the same as Alon Halevy in source 2?
- If there are uniform keys across sources, no problem.
- If not:
  – Domain specific solutions (e.g., maybe look at the address, ssn).
  – Use Information retrieval techniques (Cohen, 98).
  – Use concordance tables. These are time-consuming to build, but you can then sell them for lots of money.

Optimization and Execution
- Problem:
  – Few and unreliable statistics about the data.
  – Unexpected (possibly bursty) network transfer rates.
  – Generally, unpredictable environment.
- General solution: (research area)
  – Adaptive query processing.
  – Interleave optimization and execution. As you get to know more about your data, you can improve your plan.

Tukwila Data Integration System
- Novel components:
  – Event handler
  – Optimization-execution loop