Relational Query Optimization

Chapter 15

Overview

- Query parsing, optimization and execution
- Query evaluation plans: A Motivations example
- A typical Relational Query Optimizer
  - Translating SQL to relational algebra
  - Estimating the cost of a plan
  - Relational algebra equivalences
  - Enumeration of Alternative plans
- Other approaches

About query optimization

- Very, very, very important in DBMSes
- Commercial DBMS optimizers → trade secrets
- Typically between 40-60 person-years of development effort
- Good performance depends on the quality of the optimizer
- Aim:
  - Find a plan that is "quite good"

So what is a query plan?

- A query evaluation plan consists of an extended relational algebra tree, with annotations...
- A query is essentially treated as relational algebra select, project and join operations

Three step optimization

- Enumerate alternative plans for evaluating an expression
- Estimate the cost of the enumerated plans (use system catalog)
- Choose the plan with the lowest estimated cost.
- Two main issues:
  - For a given query, what plans are considered?
  - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?
Our Running Example: Sailing

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Joe</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>Sue</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>300</td>
<td>John</td>
<td>3</td>
<td>31</td>
</tr>
</tbody>
</table>

Sailors(sid:integer, sname: string, rating: integer, age: real)
- tuple length 50 bytes
- page holds 90 tuples, 500 pages of tuples

Reserves(sid:integer, bid:integer, day: date, name: string)
- tuple length 40 bytes
- page holds 100 tuples, we have 1000 tuples

Motivating Example

```
SELECT S.sname FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND 
    R.bid=100 AND S.rating>5
```

Reserves Sailors
- sid=sid
- bid=100
- rating > 5
- sname

RA Tree:

```
SELECT S.sname FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND 
    R.bid=100 AND S.rating>5
```

Motivating Example: The Plan

- Cost: 1000+500*1000 I/Os
- By no means the worst plan!
- Misses several opportunities: selections could have been 'pushed' earlier, no use is made of any available indexes, etc.
- Goal of optimization: To find more efficient plans that compute the same answer.

Multi-operator Queries: Pipelining versus Materialization

- Pipelining
  - The results of one operation acts as input to another, without temporary tables
- Materialization
  - The results of an operation is saved in a temporary table for processing by the next table
- Which one should be cheaper?

Creating Alternative Plans:

Two main strategies

- Pushing selects
  - reduce the size of the tables to be joined
- Using Indexes

Alternative Plan 1 (No Indexes)

- Main difference: push selects
- Assume we have 5 buffers in main memory
- Cost of plan:
  - Scan Reserves (1000) + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
  - Scan Sailors (500) + write temp T2 (250 pages, if we have 10 ratings).
  - Sort T1 (2*2*10), sort T2 (2*3*250), merge (10*250)
  - Total: 3560 page I/Os
- If we used Block Nested Join, join cost = 10*4*250, total cost = 2770.
Alternative Plan 1
(No Indexes)

- Cost of plan continued:
  - If we used Block Nested Join join, join cost = 10+4*250, total cost = 2770.
  - If we ‘push’ projections, we reduce T1 and T2
    - T1 has only sid, T2 only sid and sname:
      - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.

The bottom line: Different plans. Different costs

- Original: 50100 I/Os
- Pushing Selects: 3560 or 2770 I/Os
- Using Indexes: 1210 I/Os

Alternative Plan 2
With Indexes

- With clustered index on bid of Reserves, we get 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- INL with pipelining (outer is not materialized).
  - Joining out unnecessary fields from outer doesn’t help.
  - Decision not to push rating>5 before the join is based on availability of sid index on Sailors.
  - Cost: Selection of Reserves tuples (10 I/Os); for each, must get matching Sailors tuple (1000*1.2); total 1210 I/Os.

What a typical optimizer does

- Alternative plan consideration (equivalences)
  - Selects and cross-products can be combined into joins
  - Joins can be reordered
  - Selects and joins can be pushed ahead of joins
  - Left-Deep Plans
- Cost Estimation
  - Reading input tables
  - Writing intermediate tables
  - Sorting the final results

A typical Relational Query Optimizer

- Translating SQL to relational algebra
  - Query blocks
  - Estimating the cost of a plan
- Relational algebra equivalences
- Enumeration of Alternative plans
- Nested subqueries

Translating SQL Queries into Algebra

- Decomposition into a query block

```sql
SELECT S.sid, MIN(R.day)
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
GROUP BY S.sid
HAVING COUNT(*) > 1
```
Translating SQL Queries into Algebra

- Nested Blocks are usually treated as calls to a subroutine, made once per outer tuple. (This is an over-simplification, but serves for now.)

```sql
SELECT S.sid, MIN(R.day)
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' AND S.rating = Reference to Nested Block
GROUP BY S.sid
HAVING COUNT(*) > 1
```

```sql
SELECT MAX(S2.rating)
FROM (SELECT MAX(S2.age)
FROM Sailors S2)
```

Query Blocks: Units of Optimization

- For each block, the plans considered are:
  - All available access methods, for each reln in FROM clause.
  - All left-deep join trees (i.e., all ways to join the relations one-at-a-time, with the inner reln in the FROM clause, considering all reln permutations and join methods.)

Relational Algebra Equivalences

- A projection commutes with a selection that only uses attributes retained by the projection.
- Selection between attributes of the two arguments of a cross-product converts cross-product to a join.
- A selection on just attributes of R commutes with R \bowtie S (i.e., \sigma (R \bowtie S) \equiv \sigma (R) \bowtie S )
- Similarly, if a projection follows a join R \bowtie S, we can `push' it by retaining only attributes of R (and S) that are needed for the join or are kept by the projection.

Illustrating Equivalences

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Joe</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>Sue</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>300</td>
<td>John</td>
<td>3</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sid</th>
<th>lid</th>
<th>day</th>
<th>rname</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>01/01/07</td>
<td>Joe</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>01/02/07</td>
<td>Ann</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>01/03/07</td>
<td>Joe</td>
</tr>
</tbody>
</table>

More Equivalences

- Tools to write those equivalences
- .... Relational algebra equivalences
Illustrating Equivalences

<table>
<thead>
<tr>
<th>sid</th>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Joe</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>Sue</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>300</td>
<td>John</td>
<td>3</td>
<td>31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
<th>Sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>01/01/07</td>
<td>Joe</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>01/02/07</td>
<td>Ann</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>01/03/07</td>
<td>Joe</td>
</tr>
</tbody>
</table>

Cost estimation
- The Plan
- Result size

Cost Estimation
- For each plan considered, must estimate cost:
  - Must estimate cost of each operation in plan tree.
  - Depends on input cardinalities.
  - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins, etc.)
  - Must also estimate size of result for each operation in tree!
  - Use information about the input relations.
  - For selections and joins, assume independence of predicates.

Enumeration of Alternative Plans
- There are two main cases:
  - Single-relation plans
  - Multiple-relation plans
- For queries over a single relation, queries consist of a combination of selects, projects, and aggregate ops:
  - Each available access path (file scan / index) is considered, and the one with the least estimated cost is chosen.
  - 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).

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))

Cost Estimates for Single-Relation Plans
- Index I on primary key matches selection:
  - Cost is Height(I)+1 for a B+ tree, about 1.2 for hash index.
- Clustered index I matching one or more selects:
  - (NPages(I)+NPages(R)) * product of RF’s of matching selects.
- Non-clustered index I matching one or more selects:
  - (NPages(I)+NTuples(R)) * product of RF’s of matching selects.
- Sequential scan of file:
  - NPages(R).
- Note: Typically, no duplicate elimination on projections! (Exception: Done on answers if user says DISTINCT.)
Illustrating Reduction Factors

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Joe</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>200</td>
<td>Sue</td>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>300</td>
<td>John</td>
<td>3</td>
<td>31</td>
</tr>
</tbody>
</table>

Sailors(sid:integer, sname: string, rating: integer, age: real)
• tuple length 50 bytes
• page holds 50 tuples, 500 pages of tuples

Reserves(sid:integer, bid:integer, day: date, rname: string)
• tuple length 40 bytes
• page holds 100 tuples, we have 1000 tuples

Example

If we have an index on rating:
- (1/NKeys(I)) * NTuples(R) = (1/10) * 40000 tuples retrieved.
- Clustered index: (1/NKeys(I)) * (NPages(I)+NPages(R)) = (1/10) * (50+500) pages are retrieved. (This is the cost.)
- Unclustered index: (1/NKeys(I)) * (NPages(I)+NTuples(R)) = (1/10) * (50+40000) pages are retrieved.

If we have an index on sid:
- Would have to retrieve all tuples/pages. With a clustered index, the cost is 50+500, with unclustered index, 50+40000.

Doing a file scan:
- We retrieve all file pages (500).

Queries Over Multiple Relations

Fundamental decision in System R: only left-deep join trees are considered.
- As the number of joins increases, the number of alternative plans grows rapidly; we need to restrict the search space.
- Left-deep trees allow us to generate all fully pipelined plans.
  - Intermediate results not written to temporary files.
  - Not all left-deep trees are fully pipelined (e.g., SM join).

Enumeration of Left-Deep Plans

Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.

Enumerated using N passes (if N relations joined):
- Pass 1: Find best 1-relation plan for each relation.
- Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. (All 2-relation plans.)
- Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. (All N-relation plans.)

For each subset of relations, retain only:
- Cheapest plan overall, plus
- Cheapest plan for each interesting order of the tuples.

Enumeration of Plans (Contd.)

ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an interestingly ordered plan or an additional sorting operator.

An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
- i.e., avoid Cartesian products if possible.

In spite of pruning plan space, this approach is still exponential in the # of tables.

Cost Estimation for Multirelation Plans

Consider a query block:
- SELECT attribute list
  FROM relation list
  WHERE term1 AND ... AND termk
- 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.

Multirelation plans are built up by joining one new relation at a time.
- Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate.
Example

- **Pass 1:**
  - **Sailors:** B+ tree matches rating > 5, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
  - **Reserves:** B+ tree on bid matches bid = 500, cheapest.

  - We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation. For example, Reserves as outer. Hash index can be used to get Sailors tuples that satisfy sid = outer tuple’s sid value.

Nested Queries

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of calling nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered. The non-nested version of the query is typically optimized better.

SELECT S.sname
FROM Sailors S
WHERE EXISTS
  (SELECT *
   FROM Reserves R
   WHERE R.bid = 103
   AND S.sid = R.sid)

Equivalent non-nested query:
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
AND R.bid = 103

Summary

- Query optimization is an important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).

  - **Two parts to optimizing a query:**
    - Consider a set of alternative plans.
      - Must prune search space; typically, left-deep plans only.
      - Must estimate cost of each plan that is considered.
      - Must estimate size of result and cost for each plan node.
      - Key issues: Statistics, indexes, operator implementations.

  - **Single-relation queries:**
    - All access paths considered, cheapest is chosen.
    - Selections that match index, whether index key has all needed fields and/or provides tuples in a desired order.

  - **Multiple-relation queries:**
    - All single-relation plans are first enumerated.
      - Selections/projections considered as early as possible.
      - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
      - Next, for each 2-relation plan that is retained, all ways of joining another relation (as inner) are considered, etc.
      - At each level, for each subset of relations, only best plan for each interesting order of tuples is retained.

Example... Sailors, Clubs, Competitions

- Sailors(sid: integer, rating: integer, age: real, club: integer)
- Clubs(cid: integer, compid: integer, budget: real, status: string)
- Competition(compid: integer, code: integer, report: string)
Step 1,2,3

- Create ER model
- Create Relational tables
  - INDEXES???
- Insert data
  - Statistics
    - Sailors record is 20 bytes long; 20,000 Sailors
    - Club record is 40 bytes long; 5000 clubs
    - Competition: 2000 bytes long on average; 1000 competitions
    - Each club participate is 10 competitions on average
    - File system support 4000 byte pages
    - 12 buffer pages
    - Uniform distribution of data

Step 4

- Start using the database
- Query optimizer at work; DBA monitor use
- ?? OR Rather
  - Determine the most likely frequent queries
  - Assign INDEXES accordingly

Step 4: Query optimizer at work

- For each query
  - Estimate the best Access Plan
    - Indexes
    - Partitioning; Sorting; Hash join or Sort-merge Join
    - Iteration; File scan
  - Estimate the reduction factor of the Plan
  - Execute using the "best" plan

Options Available to Us

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Indexing</th>
<th>Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Scan</td>
<td>B+ or Hash if equality; Index in memory</td>
</tr>
<tr>
<td></td>
<td>Project</td>
<td>B+ tree leaves (if all satisfied)</td>
</tr>
<tr>
<td>Join</td>
<td>Simple nested join</td>
<td>Index nested join</td>
</tr>
</tbody>
</table>

Query

Select *
From Sailors S, Club C
Where S.cid = C.cid;

List the plans if
- Clustered hash index on cid in S.
- Clustered B+tree index on cid in S; Club is sorted on cid.

More Queries

- “Find all sailors with age = 30”
- “Find all projects with code = 20”

Question: For which query is a clustered index on the attribute most important?