Relational Query Optimization

Yanlei Diao

Slide Content Courtesy of R. Ramakrishnan, J. Gehrke, and J. Hellerstein
Overview of Query Evaluation

- **Query evaluation plan:** tree of *relational algebra* operators, with choice of algorithm for each operator.

- **Query optimization:** given a query, many plans are possible
  - Ideally, find the most efficient plan.
  - In practice, avoid worst plans in practice.
**SQL Refresher**

- **Query Semantics:**
  1. Take Cartesian product (a.k.a. cross-product) of relns in **FROM**, projecting only to those columns that appear in other clauses
  2. If a **WHERE** clause exists, apply all filters in it
  3. If a **GROUP BY** clause exists, form groups on the result
  4. If a **HAVING** clause exists, filter groups with it
  5. If an **ORDER BY** clause exists, make sure output is in right order
  6. If there is a **DISTINCT** modifier, remove duplicates
## Basics of Query Optimization

### Syntax

```sql
SELECT {DISTINCT} <list of columns>
FROM <list of relations>
{WHERE <list of "Boolean Factors">}
{GROUP BY <list of columns>}
{HAVING <list of Boolean Factors>}
{ORDER BY <list of columns>};
```

- Convert selection conditions to **conjunctive normal form (CNF):**
  - 
    
    (day<8/9/94 OR bid=5 OR sid=3 ) AND (rname= ‘Paul ’ OR sid=3)

- Interleave FROM and WHERE into an **operator tree for optimization.**
  - Query optimization largely works for Conjunctive Queries (only).

- Apply GROUP BY, HAVING, DISTINCT and ORDER BY at the end, pretty much in that order.
Outline of topics

- Query plans and equivalences
- Query optimization issues
  - Plan space
  - Cost estimation
  - Plan search
- Handling nested queries
- Multi-objective optimization in Cloud Computing
Relational Algebra Tree:

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

Expression in Relational Algebra (RA):

\[ \pi_{sname} (\sigma_{bid=100 \land rating>5} (\text{Reserves} \bowtie_{sid=sid} \text{Sailors})) \]
Query Evaluation Plan

- **Query evaluation plan** extends an RA tree with:
  1) *access method* for each relation;
  2) *implementation method* for each other operator.

- What are the missed opportunities?
  - Selections could have been `pushed’ earlier.
  - Use of indexes.
  - More efficient joins.
Relational Algebra Equivalences

- **Selections:** \( \sigma_{c_1 \land ... \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \) (Cascade)
  \[ \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \] (Commute)

- **Projections:** \( \pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_1,\ldots,a_n}(R))) \) (Cascade)

- **Joins:** \( (R \bowtie S) \equiv (S \bowtie R) \) (Commutative)
  \[ R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \] (Associative)

- Show that: \( R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S \)
More Equivalences

- $\sigma_c(R \times S) \equiv R \bowtie_c S$

- $\sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S$, if $c$ is only applied to $R$

- $\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$ holds if $\sigma$ only uses attributes retained by $\pi$

- For $\pi_b(R \bowtie_a S)$, we can ‘push’ $\pi$ before $\bowtie$ by retaining both the $a$ attribute and the $b$ attribute (if existent)

But, aggregates do not commute with other operators.
Schema for Examples

Sailors \((sid: \text{integer}, sname: \text{string}, rating: \text{integer}, age: \text{real})\)
Reserves \((sid: \text{integer}, bid: \text{integer}, day: \text{dates}, rname: \text{string})\)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
**Query Plan 1 (Selection Pushed Down)**

- **Push selections below the join.**

- **Materialization vs. Pipelining:**
  - Materialize a temporary relation T, if the next operator needs to scan T multiple times.
  - Pipelining: the opposite.

- **With 5 buffer pages, 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-Merge join: Sort T1 (2*2*10), sort T2 (2*4*250), merge (10+250).
  - Total = 4060 page I/Os.
Query Plan 2 (Different Join Method)

- Change the join method to **block nested loops join**.

With **5 buffer pages**, cost of plan:
- Scan Reserves (1000) + write temp T1 (10 pages).
- Scan Sailors (500) + write temp T2 (250 pages).
- **BNL join**: join cost = 10+4*250.
- Total cost = 2770.
Indexes

- A **tree index** matches (a conjunction of) terms if the attributes in the terms form a **prefix** of the search key.
  - Tree index on \(<a, b, c>\)
    - \(a=5 \text{ AND } b=3\) ?
    - \(a=5 \text{ AND } b>6\) ?
    - \(b=3\) ?
Query Plan 3 (Using Indexes)

- **Selection using index**: clustered index on bid of Reserves.
  - Retrieve 100,000/100 = 1000 tuples
  - Clustering: read 1000/100 = 10 pages.

- Indexed NLJ: *pipeline* the outer and *index lookup* on sid of Sailors.
  - The outer: no need to materialize.
  - The inner: sid is a key; at most one match tuple, unclustered index OK.

- Cost:
  - Selection of Reserves tuples (~10 I/Os).
  - For each tuple, get matching Sailor tuple (1000*(2~3)).
  - Total = 2010~3010 I/Os.
Outline

- Query plans and equivalences

- **Query optimization issues**
  - Cost estimation
  - Plan space
  - Plan search

- Handling nested queries

- Multi-objective optimization in Cloud Computing
An SQL query is parsed into a collection of query blocks, and these are optimized one block at a time.

Nested blocks are usually treated as calls to a subroutine, made once per outer tuple. (Optimization is advanced material...)

SELECT S.sname 
FROM   Sailors S 
WHERE  S.age IN 
       (SELECT MAX (S2.age) 
        FROM   Sailors S2 
        GROUP BY S2.rating)

Outer block   Nested block
Three Main Issues in Optimization

- Given a query block, three main optimization issues:
  - **Plan cost**: what is the cost of a given plan?
  - **Plan space**: which plans are considered?
  - **Search algorithm**: how do we search the plan space for the cheapest estimated plan?
  - We will learn the design of *System R Optimizer*
(1) Cost Estimation

- For each plan considered, must estimate its cost.

- Estimate cost of each operation in a plan tree:
  - Depends on input cardinalities.
  - Depends on the method (sequential scan, index scan, join...)

- Estimate size of result for each operation in tree:
  - Use statistics about input relations.
  - Estimate the reduction factor (RF) / selectivity of each term, which reflects the impact of the term in reducing result size.

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```
Statistics in System Catalog

- Statistics about each relation (R) and index (I):
  - **Relation cardinality**: # tuples (NTuples) in R
  - **Relation size**: # pages (NPages) in R
  - **Index cardinality**: # distinct values (NKeys) in I
  - **Index size**: # leaf pages (INPages) in I
  - **Index height**: # nonleaf levels (IHeight) of I
  - **Index range**: low/high key values (Low/High) in I
  - **Number of distinct values** in an attribute (NKeys)
  - **Histogram** for an attribute
Cost Estimates for Single-Relation Plans

- **Index I on primary key** matches selection:
  - Cost of lookup = \( \text{Height}(I) + 1 \) for a B+ tree, \( \approx 1.2 \) for hash index
  - Cost of record retrieval = 1

- **Clustered index** I matching one or more selections:
  - Cost of lookup + product of RF’s of matching terms (RF-terms) * \( \text{INPages}(I) + \text{NPages}(R) \)

- **Non-clustered index** I matching one or more selections:
  - Cost of lookup + RF-terms * INPages(I) + min(RF-terms * NTuples(R), NPages(R))

- **Sequential scan** of file: \( \text{NPages}(R) \)

- May add extra costs for GROUP BY, sorting, and duplicate elimination (if a query says DISTINCT)
Reduction Factors

- **Reduction factor (RF) or Selectivity** of each term reflects the impact of the term in reducing result size.
  - **Assumption 1**: uniform distribution of the values!
  - Term col=value: RF = 1/NKeys(I), if there is an index I on col.
  - Term col>value: RF = (High(I)-value)/(High(I)-Low(I))
  - Term $R.col1=S.col2$:
    1) If $R.col1$ is a foreign key, $S.col2$ is a primary key, then RF = 1/NTuples(S)
    2) Otherwise, RF = 1/MAX(NKeys(I1), NKeys(I2))
       - WLOG, NKeys(I1) < NKeys(I2)
       - Each value from R, which is supposed to be in the smaller index I1, has a matching value in S with the larger index I2.
       - Values in S are evenly distributed.
       - So each R tuple has NTuples(S)/NKeys(I2) matches, a RF of 1/NKeys(I2).
Illustration for $R.col1 = S.col2$

Smaller index I1

Larger index I2

$NTuples(S)$

$NKeys(I2)$
Size Estimation & Reduction Factors

- **Reduction factor (RF)** of all terms = product of all RF’s

- **Result cardinality** = max_num_tuples * product of all RF’s.
  - max_num_tuples = the product of the cardinalities of relations in the FROM clause.
  - **Assumption 2**: terms are independent!
Rethinking of Assumption 1

- “Uniform distribution of values”: often causes highly inaccurate estimates
  - E.g., distribution of gender: male (40), female (4)
  - E.g., distribution of age:
    
    | Age | Count |
    |-----|-------|
    | 0   | 2     |
    | 1   | 3     |
    | 2   | 3     |
    | 3   | 1     |
    | 4   | 2     |
    | 5   | 1     |
    | 6   | 3     |
    | 7   | 8     |
    | 8   | 4     |
    | 9   | 2     |
    | 10  | 0     |
    | 11  | 1     |
    | 12  | 2     |
    | 13  | 4     |
    | 14  | 9     |

  Nkeys = 15, count = 45.
  Reduction factor of ‘age=14’: 1/15? 9/45=1/5!

- Histogram: approximates a data distribution
Equiwidth Histograms: buckets of equal size.

Frequency: 8/3 4/3 15/3 3/3 15/3
Count: 8 4 15 3 15
Bucket: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Distribution of age:
0 (2), 1 (3), 2 (3), 3 (1), 4 (2),
5 (1), 6 (3), 7 (8), 8 (4), 9 (2),
10 (0), 11 (1), 12 (2), 13 (4), 14 (9).
Nkeys = 15, count = 45.
Reduction factor of ‘age=14’ : 9/45=1/5!
Equidepth Histograms: equal counts across buckets.

<table>
<thead>
<tr>
<th>Bucket</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Count</strong></td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>9/4</td>
<td>10/4</td>
<td>10/2</td>
<td>7/4</td>
<td>9/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>10/2</td>
<td>7/4</td>
<td>7</td>
<td>9</td>
<td>9/4</td>
<td>10/4</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>9/45=1/5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Equidepth Histograms

Small errors for infrequent items: tolerable.

Distribution of age:

0 (2), 1 (3), 2 (3), 3 (1), 4 (2),
5 (1), 6 (3), 7 (8), 8 (4), 9 (2),
10 (0), 11 (1), 12 (2), 13 (4), 14 (9).

Nkeys = 15, count = 45.

Reduction factor of ‘age=14’ : 9/45=1/5 !
**Equidepth Histograms**

Equidepth Histograms: equal counts across buckets.

<table>
<thead>
<tr>
<th>Bucket</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>9/4</td>
<td>10/4</td>
<td>10/2</td>
<td>7/4</td>
<td>9/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Favors** frequent values.
- **Representation:**
  - Boundaries of k=5 buckets {0, 4, 8, 10, 14, 14}
  - Count of tuples and number of distinct values for each bucket

Small errors for infrequent items: tolerable.

Now accurate for value 14: 9/45=1/5
Equidepth Histograms: equal counts across buckets.

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Frequency</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9/4</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>10/4</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10/2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>7/4</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>9/1</td>
<td>9</td>
</tr>
</tbody>
</table>

Small errors for infrequent items: tolerable.

Now accurate for value 14: 9/45 = 1/5

- Algorithm to build a k-bucket histogram over R.a
  - (Collect a sample of size m from R.a, e.g., reservoir sampling)
  - Sort the original column or the sample by R.a
  - Break the sorted list by equal count m/k, and find boundary values
Reservoir Sampling

Example: Sample size 10

Suppose we see a sequence of items, one at a time. We want to keep ten items in memory, and we want them to be selected at random from the sequence. If we know the total number of items \( n \), then the solution is easy: select ten distinct indices \( i \) between 1 and \( n \) with equal probability, and keep the \( i \)-th elements. The problem is that we do not always know \( n \) in advance. A possible solution is the following:

- Keep the first ten items in memory.
- When the \( i \)-th item arrives (for \( i > 10 \)):
  - with probability \( 10/i \), keep the new item (discard an old one, selecting which to replace at random, each with chance \( 1/10 \))
  - with probability \( 1 - 10/i \), keep the old items (ignore the new one)

So:

- when there are 10 items or fewer, each is kept with probability 1;
- when there are 11 items, each of them is kept with probability \( 10/11 \); for the old items, that is \( (1)(1/11 + (10/11)(9/10)) = 1/11 + 9/11 = 10/11 \)
- when there are 12 items, the twelfth item is kept with probability \( 10/12 \), and each of the previous 11 items are also kept with probability \( (10/11)(2/12 + (10/12)(9/10)) = (10/11)(11/12) = 10/12 \);
- by induction, it is easy to prove that when there are \( n \) items, each item is kept with probability \( 10/n \).
Rethinking Assumption 2

- “Independence of predicates”: causes inaccurate estimates
  - E.g., Car DB: 10 makes, 100 models.
  - RF of make=‘honda’ and model=‘civic’
  - If independent, 1/10 * 1/100. In practice, much higher!

- Multi-dimensional histograms [PI’97, MVW’98, GKT’00]
  - Maintain counts and frequency in multi-attribute space.

- Dependency-based histograms [DGR’01]
  - Learn dependency between attributes and compute conditional probability $P(\text{model} = \text{‘civic’} \mid \text{make} = \text{‘honda’})$
  - Can use graphical models…
(2) Plan Space

- For each query block, the plans considered are:
  - All *access methods*, for each reln in the FROM clause.
  - All *left-deep join trees*: all the ways to join the relns one-at-a-time, with the inner reln in the FROM clause.
    - Number of left-deep join trees for N relns?
    - All permutations of N relns: \( N! \)!
Plan Space

- For each block, the plans considered are:
  - All access methods, for each reln in FROM clause.
  - All left-deep join trees: all the ways to join the relns one-at-a-time, with the inner reln in the FROM clause.
    - All permutations of N relns: N factorial!
    - But avoid Cartesian products!
    - Join R, S, T with R.a = S.a and S.b = T.b. How many left-deep trees are valid?
  - All join methods, for each join in the tree.
  - Appropriate places for selections and projections.
(3) Plan Search

- As the number of joins increases, the number of alternative plans grows rapidly.

- System R: (1) use only left-deep join trees, (2) avoid Cartesian products.
  - Motivation: allow pipelined plans; intermediate results not written to temp files.
  - Not all left-deep trees are fully pipelined!
    - Sort-Merge join: at least sorting phase
    - Two-phase hash join: partitioning phase
**Search Algorithm**

- Left-deep join plans:
  - Differ in the *order* of relations, *access method* for each relation, *join method* for each join.
  - But may share **common prefixes**. Don’t enumerate all. Instead use…

**Dynamic Programming**

“a method for solving problems that exhibit the properties of *overlapping subproblems* and *optimal substructures*”

- What are the overlapping subproblems?
- What do optimal substructures mean?
**An Example Star Schema**

- **Dynamic Programming**
  
  “a method for solving problems that exhibit the properties of *overlapping subproblems* and *optimal substructures*”

- Find the best plans to access A, B, C, D individually
- Repeat this for 4 relation sets: join (A-B-C)-D, (A-B-D)-C, (A-B-D)-C; store the best for (A-B-C-D)

① This procedure is restricted by join predicates of A,B,C,D (i.e., left deep trees but avoiding Cartesian products).  
② Number of plans enumerated ≠ Number of possible plans (N!)
System R: Enumeration of Left-Deep Plans

- Enumerate with N passes (if N relations are joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best ways to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - ...
  - Pass N: Find best ways 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 *unordered* plan, and
  - cheapest plan for each *interesting order* (order for final output or a subsequent op. using sorting) of the tuples.
A $k$-way ($k<N$) plan is not combined with an additional relation unless there is a join condition between them.

- Do it until all predicates in WHERE have been used up.
- That is, avoid cartesian products if possible.

ORDER BY, GROUP BY, aggregates etc. handled as a final step, using an `interestingly ordered’ plan, or an additional sorting, or hashing.
**Complexity of Plan Search**

- Enumeration of all left-deep plans for an n-way join: \( O(n!) \), where \( n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \) with a large \( n \).

- System R yields a better cost: consider a star join graph
  - \( R.a_1 = S_1.a_1 \)
  - \( R.a_2 = S_2.a_2 \)
  - \( ... \)
  - \( R.a_{n-1} = S_{n-1}.a_{n-1} \)

- Total number of plans considered?
- Max. number of plans stored in an intermediate pass?
explain analyze
select paperid, authid
from paperauths, authors
where authors.id=authid;

QUERY PLAN
--------------------------------------------------------
Hash Join  (cost=55249.17..396912.22 rows=8997842 width=8)
  (actual time=563.063..4223.015 rows=8997872 loops=1)
  Hash Cond: (paperauths.authid = authors.id)
  ->  Seq Scan on paperauths  (cost=0.00..129792.42 rows=8997842 width=8)
      (actual time=0.035..645.836 rows=8997872 loops=1)
  ->  Hash  (cost=27500.41..27500.41 rows=1691341 width=4)
      (actual time=561.896..561.896 rows=1691341 loops=1)
      Buckets: 131072  Batches: 32  Memory Usage: 2895kB
      ->  Seq Scan on authors  (cost=0.00..27500.41 rows=1691341 width=4)
          (actual time=0.025..265.960 rows=1691341 loops=1)
Planning time: 1.516 ms
Execution time: 4461.836 ms
(8 rows)
Estimating Query Latency

Figure 3: A neural network for a simple join query
Overview of Cloud Computing
Multi-Objective Optimization
Multi-Objective Optimization

- MOO still has the substructure property, but loses the suboptimality problem.
  - See Example 1 in “Approximation Schemes for Many-Objective Query Optimization”
- As a result, the number of possible plans = the number of plans enumerated
- The complexity depends on the Catalan number.
  - See Section 5.2