CS 4604: Introduction to Database Management Systems

Query Optimization

Virginia Tech CS 4604 Sprint 2021
Instructor: Yinlin Chen
Today’s Topics

• Query Optimization
Usually there is a heuristics-based rewriting step before the cost-based steps.
Query Parsing & Optimization

• Query parser
  • Check correctness, authorization
  • Generates a parse tree
  • Straightforward

• Query rewriter
  • Converts queries to canonical form
    • flatten views
    • subqueries into fewer query blocks
  • Weak spot in many open-source DBMSs
Query Parsing & Optimization

• “Cost-based” Query Optimizer
  • Optimizes 1 query block at a time
    • Select, Project, Join
    • GroupBy/Agg
    • Order By (if top-most block)
  • Uses catalog stats to find least-“cost” plan per query block
  • “Soft underbelly” of every DBMS
    • Sometimes not truly “optimal”

Query Parser

Select *
From Blah B
Where B.blah = blah

Query Rewriter

Query Optimizer

Catalog Manager

Schema & Statistics

Plan Generator

Plan Cost Estimator

Query Plan Executor

VT VIRGINIA TECH
Query Optimization Overview

- Query block can be converted to relational algebra
- Relational algebra converts to tree
- Each operator has implementation choices
- Operators can also be applied in different orders!

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

\[ \pi_{\text{sname}} \sigma_{\text{bid}=100 \land \text{rating} > 5} (\text{Reserves} \Join \text{Sailors}) \]
Query Optimization: The Components

• Three beautifully orthogonal concerns:
  – Plan space:
    • for a given query, what plans are considered?
  – Cost estimation:
    • how is the cost of a plan estimated?
  – Search strategy:
    • how do we “search” in the “plan space”?
Query Optimization: The Goal

• Optimization goal:
  – Ideally: Find the plan with least actual cost.
  – Reality: Find the plan with least estimated cost.
    • And try to avoid really bad actual plans!
Query Optimization: Example

Canonical Form has the following properties:
1. Push Selections as much as possible.
2. Push Projections as much as possible.
3. It is a left-deep join tree (we will see this later).
Relational Algebra Equivalences

• Selections:
  - \( \sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots(\sigma_{c_n}(R))\ldots) \) (cascading)
  - \( \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \) (commutative)

• Projections:
  - \( \pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_1}, \ldots, \pi_{a_{n-1}}(R))\ldots) \) (cascading)
Relational Algebra Equivalences

• Cartesian Product
  – $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  – $R \times S \equiv S \times R$ (commutative)

• Join
  – $R \Join (S \Join T) \equiv (R \Join S) \Join T$ (associative)
  – $R \Join S \equiv S \Join R$ (commutative)
Are Joins Associative and Commutative?

• After all, just Cartesian Products with Selections
• You can think of them as associative and commutative...
• ...But beware of join turning into cross-product!
  – Consider R(a,z), S(a,b), T(b,y)

```
SELECT *
FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b;
```

– \((S \bowtie_{b=b} T) \bowtie_{a=a} R \not\equiv S \bowtie_{b=b} (T \bowtie_{a=a} R)\) *(not legal!!)*
– \((S \bowtie_{b=b} T) \bowtie_{a=a} R \not\equiv S \bowtie_{b=b} (T \times R)\) *(not the same!!)*
– \((S \bowtie_{b=b} T) \bowtie_{a=a} R \equiv S \bowtie_{b=b \land a=a} (T \times R)\) *(the same!!)*
Join Ordering

• Similarly, note that some join orders have cross products, some don’t
• Equivalent for the query above:

```
SELECT *
FROM R, S, T
WHERE R.a = S.a
AND S.b = T.b;
```
1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

\[ \sigma_{\theta_1 \wedge \theta_2} (E) = \sigma_{\theta_1} (\sigma_{\theta_2} (E)) \]

2. Selection operations are commutative.

\[ \sigma_{\theta_1} (\sigma_{\theta_2} (E)) = \sigma_{\theta_2} (\sigma_{\theta_1} (E)) \]

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

\[ \Pi_{L_1} (\Pi_{L_2} (\ldots (\Pi_{L_n} (E))\ldots)) = \Pi_{L_1} (E) \]

4. Selections can be combined with Cartesian products and theta joins.
   
a. \[ \sigma_\theta (E_1 \times E_2) = E_1 \bowtie_\theta E_2 \]
   
b. \[ \sigma_{\theta_1} (E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2 \]
5. Theta-join operations (and natural joins) are commutative. 
\[ E_1 \Join_\theta E_2 = E_2 \Join_\theta E_1 \]

6. (a) Natural join operations are associative: 
\[ (E_1 \Join E_2) \Join E_3 = E_1 \Join (E_2 \Join E_3) \]

(b) Theta joins are associative in the following manner: 
\[ (E_1 \Join_{\theta_1} E_2) \Join_{\theta_2 \land \theta_3} E_3 = E_1 \Join_{\theta_1 \land \theta_3} (E_2 \Join_{\theta_2} E_3) \]

where \( \theta_2 \) involves attributes from only \( E_2 \) and \( E_3 \).
7. The selection operation distributes over the theta join operation under the following two conditions:
   (a) When all the attributes in $\theta_0$ involve only the attributes of one of the expressions ($E_1$) being joined.

   \[ \sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2 \]

   (b) When $\theta_1$ involves only the attributes of $E_1$ and $\theta_2$ involves only the attributes of $E_2$.

   \[ \sigma_{\theta_1 \land \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2)) \]
Some Common Heuristics: Selections

• Selection cascade and pushdown
  – Apply selections as soon as you have the relevant columns
  – Ex:
    • $\pi_{\text{sname}} (\sigma_{\text{bid}=100 \land \text{rating} > 5} (\text{Reserves} \bowtie_{\text{sid} = \text{sid}} \text{Sailors}))$
    • $\pi_{\text{sname}} (\sigma_{\text{bid}=100} (\text{Reserves}) \bowtie_{\text{sid} = \text{sid}} \sigma_{\text{rating} > 5} (\text{Sailors}))$
Some Common Heuristics: Projections

- Projection cascade and pushdown
  - Keep only the columns you need to evaluate downstream operators
  - Ex:
    - $\pi_{\text{sname}} \sigma_{\text{bid}=100 \land \text{rating} > 5} (\text{Reserves} \bowtie_{\text{sid} = \text{sid}} \text{Sailors})$
    - $\pi_{\text{sname}} (\pi_{\text{sid}} (\sigma_{\text{bid}=100} (\text{Reserves})) \bowtie_{\text{sid} = \text{sid}} \pi_{\text{sname}, \text{sid}} (\sigma_{\text{rating} > 5} (\text{Sailors})))$
Some Common Heuristics

- Avoid Cartesian products
  - Given a choice, do theta-joins rather than cross-products
  - Consider $R(a,b)$, $S(b,c)$, $T(c,d)$
  - Favor $(R \bowtie S) \bowtie T$ over $(R \times T) \bowtie S$
Query Parsing & Optimization

Select * 
From Blah B 
Where B.blah = blah 

Query Parser 

Query Rewriter 

Query Optimizer 

Plan Generator 

Plan Cost Estimator 

Catalog Manager 

Schema & Statistics 

Query Plan Executor 

Usually there is a heuristics-based rewriting step before the cost-based steps.
Schema for Examples

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

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats

- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings

- Assume we have 5 pages to use for joins.
Motivating Example: Plan 1

- Here’s a reasonable query plan:

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

On-the-fly

![Diagram of query plan]

Sailors SCAN

Reserves SCAN

PAGE NESTED LOOPS
Motivating Example: Plan 1 Cost

- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- For each page of Sailors, Scan Reserves (1000 IOs)
- Total: $500 + 500 \times 1000$
  - 500,500 IOs
- Bad plan!
- Goal of optimization:
  - Find less cost (faster) plan that compute the same answer
Plan 2: Selection Pushdown

500,500 IOs
Plan 2 Cost Analysis

- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors, Scan Reserves (1000 IOs)
- Total: \(500 + 250 \times 1000 = 250,500\) IOs
Plan 3: More Selection Pushdown

250,500 IOs
Plan 3 Cost Analysis

• Let’s estimate the cost:
• Scan Sailors (500 IOs)
• For each pageful of high-rated Sailors, Scan Reserves (1000 IOs)
• Total: $500 + 250 \times 1000 = 250,500$ IOs
More Selection Pushdown Analysis

Pushing a selection into the inner loop of a nested loop join doesn't save I/Os! Essentially equivalent to having the selection above.

\[ \pi_{\text{sname}} \]
\[ \sigma_{\text{bid}=100} \]
\[ \bowtie_{\text{sid}=\text{sid}} \]
\[ \text{PAGE NESTED LOOPS} \]
\[ \sigma_{\text{rating} > 5} \]
\[ \text{Sailors SCAN} \]
\[ \text{Reserves SCAN} \]

250,500 I/Os

\[ \pi_{\text{sname}} \]
\[ \bowtie_{\text{sid}=\text{sid}} \]
\[ \text{PAGE NESTED LOOPS} \]
\[ \sigma_{\text{rating} > 5} \]
\[ \text{Sailors SCAN} \]
\[ \text{Reserves SCAN} \]
\[ \sigma_{\text{bid}=100} \]

250,500 I/Os
Plan 4: Join Ordering

250,500 IOs
Plan 4 Cost Analysis

• Let’s estimate the cost:
• Scan Reserves (1000 IOs)
• For each pageful of Reserves for bid 100,
  Scan Sailors (500 IOs)
• Total: \(1000 + 10 \times 500 = 6000\) IOs
Plan 5: Materializing Inner Loops

$\pi_{\text{name}}$

$\neg\neg_{\text{sid} = \text{sid}}$

$\sigma_{\text{bid} = 100}$

Reserves SCAN

$\sigma_{\text{rating} > 5}$

Sailors SCAN

$\neg\neg_{\text{sid} = \text{sid}}$

$\sigma_{\text{bid} = 100}$

Reserves SCAN

$\sigma_{\text{rating} > 5}$

Sailors SCAN

6000 IOs
Plan 5 Cost Analysis

- Let’s estimate the cost:
- Scan Reserves (1000 IOs)
- Scan Sailors (500 IOs)
- Materialize Temp table T1 (250 IOs)
- For each pageful of Reserves for bid 100,
  Scan T1 (250 IOs)
- Total: 1000 + 500 + 250 + (10 * 250) = 4250 IOs
Plan 6: Join Ordering Again

\[
\pi_{\text{sname}} \left\langle \sigma_{\text{bid}=100} \left( \text{Reserves SCAN} \right) \right\rangle \left\langle \sigma_{\text{rating} > 5} \left( \text{Sailors SCAN} \right) \right\rangle \left\langle \sigma_{\text{sid} = \text{sid}} \text{PAGE NESTED LOOPS} \right\rangle \text{mat} \]

\[
\pi_{\text{sname}} \left\langle \sigma_{\text{rating} > 5} \left( \text{Sailors SCAN} \right) \right\rangle \left\langle \sigma_{\text{bid}=100} \left( \text{Reserves SCAN} \right) \right\rangle \left\langle \sigma_{\text{sid} = \text{sid}} \text{PAGE NESTED LOOPS} \right\rangle \text{mat} \]

4250 IOs
Plan 6 Cost Analysis

- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- Scan Reserves (1000 IOs)
- Materialize Temp table T1 (10 IOs)
- For each pageful of high-rated Sailors, Scan T1 (10 IOs)
- Total: $500 + 1000 + 10 + (250 \times 10) = 4010$ IOs
Plan 7: Join Algorithm

4010 IOs
Plan 7 Cost Analysis

- With 5 buffers, cost of plan:
  - Scan Reserves (1000)
  - Scan Sailors (500)
- Sort high-rated sailors
  Note: pass 0 doesn’t do read I/O, just gets input from select.
- Sort reservations for boat 100
  Note: pass 0 doesn’t do read I/O, just gets input from select.
- Merge (10+250) = 260
- Total: sum above
Plan 7 Cost Analysis

- With 5 buffers, cost of plan:
  - Scan Reserves (1000)
  - Scan Sailors (500)
- Sort reservations for boat 100
  - 2 passes for reserves
    - pass 0 = 10 to write, pass 1 = 2*10 to read/write
- Sort high-rated sailors
  - 4 passes for sailors
    - pass 0 = 250 to write, pass 1,2,3 = 2*250 to read/write
- Merge (10+250) = 260

1000 + 500 + sort reserves (10 + 2*10* 1) + sort sailors (250 + 2*250*3) + merge (10+250) = 3540 IOs
Join Algorithm and Materializing Inner Loops

3540 IOs
Plan 8 Cost Analysis

- With 5 buffers, cost of plan:
  - Scan Sailors (500), write T1 (250)
  - Scan Reserves (1000), write T2 (10)
  - Sort T1
  - Sort T2

- How many passes for each sort?
  - 2 passes for reserves ($2 \times 10 \times 2$ to read/write)
  - 4 passes for sailors ($2 \times 250 \times 4$ to read/write)

- Merge (10+250) = 260

- Total:
  1000 + 500 + 10 + 250 + 2*10*2 +
  $2 \times 250 \times 4$ + merge (10+250) = 4060 IOs
Another Join Algorithm

\[ \pi_{\text{sname}} \]

\[ \bowtie_{\text{sid}=\text{sid}} \]

\[ \sigma_{\text{rating} > 5} \]

\[ \text{Sailors} \]
\[ \text{SCAN} \]

\[ \sigma_{\text{bid}=100} \]

\[ \text{Reserves} \]
\[ \text{SCAN} \]

\[ \text{mat} \]

\[ 4010 \text{ IOs} \]
Plan 9 Cost Analysis

- With 5 buffers, cost of plan:
  - Scan Sailors (500)
  - Scan Reserves (1000)
  - Write Temp T1 (10)
  - For each blockful of high-rated sailors
    - Loop on T1 (\(\lceil \frac{S}{B-2} \rceil \times [T]\))
  - Total:

\[
500 + 1000 + 10 + (\text{ceil}(250/3) \times 10) = 500 + 1000 + 10 + (84 \times 10) = 2350 \text{ IOs}
\]
How About Indexes?

• Indexes:
  – Reserves.bid clustered
  – Sailors.sid unclustered

• Assume indexes fit in memory

Reserves: bid

\[ \text{bid} = 100 \text{ (on 10 pages)} \]
Index Cost Analysis

- **No projection pushdown to left for** $\pi_{\text{snname}}$
  - Projecting out unnecessary fields from outer of Index NL doesn’t make an I/O difference.

- **No selection pushdown to right for** $\sigma_{\text{rating} > 5}$
  - Does not affect Sailors.sid index lookup

- With clustered index on bid of Reserves, we access how many pages of Reserves?:
  - $100,000/100 = 1000$ tuples on $1000/100 = 10$ pages.

- Join column sid is a **key** for Sailors.
  - At most one matching tuple, unclustered index on sid OK

1010 IOs
Index Cost Analysis Part 2

- With clustered index on bid of Reserves, we access how many pages of Reserves?:
  - $100,000/100$ (boats) = $1000$ tuples on $1000/100 = 10$ pages.

- for each Reserves tuple $1000$
  - get matching Sailors tuple (1 IO)
  - (recall: 100 Reserves per page, 1000 pages)

- $10 + 1000 \times 1 = 1010$ IOs

- Cost: Selection of Reserves tuples (10 I/Os); then, for each, must get matching Sailors tuple (1000); total 1010 I/Os.
Summing up

• There are *lots* of plans
  – Even for a relatively simple query

• Not so clear that’s true!
  – Manual query planning can be tedious, technical
  – Machines are better at enumerating options than people
    • Hence AI
  – We will see soon how optimizers make simplifying assumptions
Query Optimization

- Given: A closed set of operators
  - Relational ops (table in, table out)
  - Physical implementations (of those ops and a few more)

- Plan space
  - Based on relational equivalences, different implementations

- Cost Estimation based on
  - Cost formulas
  - Size estimation, in turn based on
    - Catalog information on base tables
    - Selectivity (Reduction Factor) estimation

- A search algorithm
  - To sift through the plan space and find lowest cost option!
A Naïve Query Optimizer

• Given an input query Q:
  1. Enumerate all possible plans for Q
     • Too many plans to consider!
  2. Estimate the cost of each plan
     • Hard to estimate cost accurately given caches etc.
  3. Pick plan with the lowest cost
     • How? Keep all plans in memory?
     • What if there are million alternative ways of executing the Q?
The System R Optimizer

• Plan Space
  – Many plans have the same high cost subtree that can be pruned
  – Heuristics(aka tricks that usually work):
    • Consider only left-deep plans
    • Avoid Cartesian products
    • Don’t optimize the entire query at once

• Cost estimation
  – Inexact is fine as long as we can compare plans
    • Better estimators have been developed

• Search Algorithm
  – Dynamic Programming
Query Optimization

1. Plan Space
2. Cost Estimation
3. Search Algorithm
Query Blocks: Units of Optimization

• Break query into query blocks
• Optimize one block at a time
• Uncorrelated nested blocks computed once
• Correlated nested blocks are like function calls
  – But sometimes can be “decorrelated”
  – Recall relational algebra lecture

```
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**
Query Blocks: Units of Optimization

• For each block, the plans considered are:
  – All relevant access methods, for each relation in FROM clause
  – All left-deep join trees
    • right branch always a base table
    • consider all join orders and join methods

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

Sailors (sid: integer, sname: text, rating: integer, age: float)

Reserves (sid: integer, bid: integer, day: date, rname: text)

- **Reserves:**
  - Each tuple is 40 bytes long,
  - 100 tuples per page, 1000 pages.
  - 100 distinct bids.

- **Sailors:**
  - Each tuple is 50 bytes long,
  - 80 tuples per page, 500 pages.
  - 10 ratings, 40,000 sids.
“Physical” Properties

- Two common “physical” properties of an output:
  - Sort order
  - Hash Grouping

- Certain operators produce these properties in output
  - E.g., Index scan (result is sorted)
  - E.g., Sort (result is sorted)
  - E.g., Hash (result is grouped)

- Certain operators require these properties at input
  - E.g., MergeJoin requires sorted input

- Certain operators preserve these properties from inputs
  - E.g., MergeJoin preserves sort order of inputs
  - E.g., Index nested loop join (INLJ) preserves sort order of outer (left) input
Physically Equivalent Plans

- Same content and same physical properties
Queries Over Multiple Relations

• A System R heuristic: only left-deep join trees considered
  – Restricts the search space
  – Left-deep trees allow us to generate all fully pipelined plans
    • i.e., intermediate results not written to temporary files
    • Not all left-deep trees are fully pipelined (e.g., SM join).

![Diagram of tree structures]

- Left-deep tree
- Linear tree
- Bushy tree
Plan Space Review

- For a SQL query, full plan space:
  - All equivalent relational algebra expressions
    - Based on the equivalence rules we learned
  - All mixes of physical implementations of those algebra expressions

- We might prune this space:
  - Selection/Projection pushdown
  - Left-deep trees only
  - Avoid Cartesian products

- Along the way we may care about physical properties like sorting
  - Because downstream ops may depend on them
  - And enforcing them later may be expensive
Query Optimization

1. Plan Space

2. Cost Estimation

3. Search Algorithm
Cost Estimation

• For each plan considered, must estimate total cost:
  – Must estimate cost of each operation in plan tree
    • Depends on input cardinalities.
    • sequential scan, index scan, joins, etc.

• Must estimate size of result for each operation in tree!
  – Because it determines downstream input cardinalities!
  – Use information about the input relations.
  – For selections and joins, assume independence of predicates.

• In System R, cost is boiled down to a single number consisting of #I/O + CPU-factor * #tuples
  – Second term estimate the cost of tuple processing
Statistics and Catalogs

- Need info on relations and indexes involved.
- **Catalogs** typically contain at least:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTuples</td>
<td># of tuples in a table (cardinality)</td>
</tr>
<tr>
<td>NPages</td>
<td># of disk pages in a table</td>
</tr>
<tr>
<td>Low/High</td>
<td>min/max value in a column</td>
</tr>
<tr>
<td>Nkeys</td>
<td># of distinct values in a column</td>
</tr>
<tr>
<td>IHeight</td>
<td>the height of an index</td>
</tr>
<tr>
<td>INPages</td>
<td># of disk pages in an index</td>
</tr>
</tbody>
</table>

- Catalogs updated periodically.
  - Too expensive to do continuously
  - Lots of approximation anyway, so a little slop here is ok.
- Modern systems do more
  - Especially keep more detailed statistical information on data values. e.g., histograms
Size Estimation and Selectivity

• Max output cardinality = product of input the cardinalities of the relations in FROM

• Selectivity (sel) associated with each term in WHERE
  – Reflects the impact of the term in reducing result size.
  – Selectivity = |output| / |input|
  – Selectivity: “Reduction Factor” (RF)
  – Always between 0 and 1

```
SELECT  attribute list
    FROM  relation list
    WHERE  term1 AND ... AND termk
```
Result Size Estimation

• Result cardinality = Max # tuples * \textit{product} of all selectivities.

• Term col=value (given Nkeys(col) unique values of col)
  – sel = 1/NKeys(col)

• Term col1=col2 (handy for joins too...)
  – sel = 1/MAX(NKeys(col1), NKeys(col2))

• Term col>value
  – sel = (High(col)-value)/(High(col)-Low(col))

• Term in
  – sel = 1/NKeys(col) * # items in the list
/ * Note: the default selectivity estimates are not chosen entirely at random. * We want them to be small enough to ensure that indexscans will be used if * available, for typical table densities of ~100 tuples/page. Thus, for * example, 0.01 is not quite small enough, since that makes it appear that * nearly all pages will be hit anyway. Also, since we sometimes estimate * eqsel as 1/num_distinct, we probably want DEFAULT_NUM_DISTINCT to equal * 1/DEFAULT_EQ_SEL. */

/* default selectivity estimate for equalities such as "A = b" */
#define DEFAULT_EQ_SEL 0.005

/* default selectivity estimate for inequalities such as "A < b" */
#define DEFAULT_INEQ_SEL 0.3333333333333333

/* default selectivity estimate for range inequalities "A > b AND A < c" */
#define DEFAULT_RANGE_INEQ_SEL 0.005

/* default selectivity estimate for multirange inequalities "A > b AND A < c" */
#define DEFAULT_MISTRANGE_INEQ_SEL 0.005

/* default selectivity estimate for pattern-match operators such as LIKE */
#define DEFAULT_MATCH_SEL 0.005

/* default selectivity estimate for other matching operators */
#define DEFAULT_MATCHING_SEL 0.010

/* default number of distinct values in a table */
#define DEFAULT_NUM_DISTINCT 200

/* default selectivity estimate for boolean and null test nodes */
#define DEFAULT_UNK_SEL 0.005
#define DEFAULT_NOT_UNK_SEL (1.0 - DEFAULT_UNK_SEL)
Reduction Factors & Histograms

Distribution D

Uniform distribution approximating D
Reduction Factors & Histograms

Equiwidth histogram

Equidepth histogram ~ quantiles
Selectivity Example: Join Selectivity

\[ R \bowtie_p \sigma_q(S) \]

algebraic equivalence: \[ R \bowtie_p S \equiv \sigma_p(R \times S) \]

Join selectivity is selectivity \( s_p \) \quad \text{Total rows:} \quad s_p \times |R| \times |S| \n
\[ R \bowtie_p \sigma_q(S) \equiv \sigma_p(R \times \sigma_q(S)) \equiv \sigma_{p \land q}(R \times S) \]

Join selectivity is selectivity \( s_p s_q \) \quad \text{Total rows:} \quad s_p s_q \times |R| \times |S|
Selectivity Example: Column Equality

T.p = T.age ??

Idea: scan over all values of p and age, and check when they are equal
Selectivity Example: Column Equality

T.p = T.age ??
Idea: scan over all values of p and age, and check when they are equal

\[
T.p = T.age \\
= (T.p = 40 \land T.age = 40) \lor (T.p = 41 \land T.age = 41) \lor (T.p = 42 \land T.age = 42) \ldots \\
= (T.p = 40 \land T.age = 40) + (T.p = 41 \land T.age = 41) + (T.p = 42 \land T.age = 42) \ldots \\
= (T.p = 40 * T.age = 40) + (T.p = 41 * T.age = 41) + (T.p = 42 * T.age = 42) \ldots
\]

\[
\begin{aligned}
& (T.p = 40) \\
& \frac{\text{height}(\text{bin}_p(40))}{\text{width}(\text{bin}_p(40)) \times n}
\end{aligned} \quad \begin{aligned}
& (T.age = 40) \\
& \frac{\text{height}(\text{bin}_{age}(40))}{\text{width}(\text{bin}_{age}(40)) \times n}
\end{aligned}
\]

Independence assumption

Uniform assumption

Just add up all the values...
Compute Selectivities

• Know how to compute selectivities for basic predicates
  – The System R version
  – The histogram version
• Assumption 1: uniform distribution within histogram bins
  – Within a bin, fraction of range = fraction of count
• Assumption 2: independent predicates
  – Selectivity of AND = product of selectivities of predicates
  – Selectivity of OR = sum of selectivities of predicates - product of selectivities of predicates
  – Selectivity of NOT = 1 – selectivity of predicates
• Joins are not a special case
  – Simply compute the selectivity of all predicates
  – And multiply by the product of the table sizes
Summary: Selectivity Estimation

- We need a way to estimate the size of the intermediate tables
  Recall cost of each operator = I/Os (to bring in input) + CPU-factor * # tuples processed
- Output size = input size * operator selectivity

**System R**
- col=value
  - 1/uniq-keys(col)
- col1=col2
  - 1/\(\text{MAX}(\text{uniq-keys}(\text{col1}), \text{uniq-keys}(\text{col2}))\)
- col>value
  \[ \frac{\text{High}(\text{col}) - \text{value}}{\text{High}(\text{col}) - \text{Low}(\text{col}) + 1} \]

**Histogram**
- col=value
  - bar height containing value
    \[ \text{# values contained in bar} \]
- col1=col2
  - Breakdown into
    \[ (\text{col1} = v1 \land \text{col2} = v1) \lor (\text{col1} = v2 \land \text{col2} = v2) \lor \ldots \]
- col>value
  \[ \frac{\text{sum of bar heights} > \text{value}}{\text{total number of rows}} \]
Summary: Selectivity Estimation

• In both cases, for more complex predicates:
  – \( p_1 \land p_2 \)
    • \( \text{selectivity}(p_1) \times \text{selectivity}(p_2) \)
  – \( p_1 \lor p_2 \)
    • \( \text{selectivity}(p_1) + \text{selectivity}(p_2) - (\text{selectivity}(p_1) \times \text{selectivity}(p_2)) \)
    • Last term is 0 if \( p_1 \) and \( p_2 \) are non-overlapping (e.g., \( \text{age} > 60 \) OR \( \text{age} < 21 \))
  – Not \( p_1 = 1 \) – \( \text{selectivity}(p_1) \)
Query Optimization

1. Plan Space

2. Cost Estimation

3. Search Algorithm
Enumeration of Alternative Plans

• There are two main cases:
  – Single-table plans (base case)
  – Multiple-table plans (induction)

• Single-table queries include selects, projects, and GroupBy/aggregation:
  – Consider each available access path (file scan / index)
    • Choose the one with the least estimated cost
  – Selection/Projection done on the fly
  – Result pipelined into grouping/aggregation
Cost Estimates for Single-Relation Plans

• Index I on primary key matches selection:
  – Cost is \( (\text{Height}(I) + 1) + 1 \) for a B+ tree.

• Clustered index I matching selection:
  – \( (\text{NPages}(I) + \text{NPages}(R)) \times \text{selectivity} \).

• Non-clustered index I matching selection:
  – \( (\text{NPages}(I) + \text{NTuples}(R)) \times \text{selectivity} \).

• Sequential scan of file:
  – \( \text{NPages}(R) \).

• Recall: Must also charge for **duplicate elimination** if required
Example

• If we have an index on rating:
  – **Cardinality** = \( \frac{1}{N_{\text{Keys}(I)}} \times N_{\text{Tuples}(R)} = \frac{1}{10} \times 40000 \) tuples
  – **Clustered index**: \( \frac{1}{N_{\text{Keys}(I)}} \times (N_{\text{Pages}(I)}+N_{\text{Pages}(R)}) \)
    = \( \frac{1}{10} \times (50+500) = 55 \) pages are retrieved. (This is the cost.)
  – **Unclustered index**: \( \frac{1}{N_{\text{Keys}(I)}} \times (N_{\text{Pages}(I)}+N_{\text{Tuples}(R)}) \)
    = \( \frac{1}{10} \times (50+40000) = 4005 \) 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).

```
SELECT S.sid
FROM Sailors S
WHERE S.rating=8
```
Enumeration of Left-Deep Plans

- Left-deep plans differ in
  - the order of relations
  - the access method for each leaf operator
  - the join method for each join operator

- Enumerated using N passes (if N relations joined):
  - **Pass 1**: Find best 1-relation plan for each relation
  - **Pass i**: Find best way to join result of an \((i -1)\)-relation plan (as outer) to the \(i^{th}\) relation. (\(i\) between 2 and N.)

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

• Bellman ’57 (slightly adapted to our setting)
• The best overall plan is composed of best decisions on the subplans
  – Optimal result has optimal substructures
• For example, the best left-deep plan to join tables A, B, C is either:
  – (The best plan for joining A, B) \( \bowtie \) C
  – (The best plan for joining A, C) \( \bowtie \) B
  – (The best plan for joining B, C) \( \bowtie \) A
• This is great!
  – When optimizing a subplan (e.g. A \( \bowtie \) B), we don’t have to think about how it will be used later (e.g. when dealing with C)!
  – When optimizing a higher-level plan (e.g. A \( \bowtie \) B \( \bowtie \) C) we can reuse the best results of subroutines (e.g. A \( \bowtie \) B)!
Dynamic Programming Algorithm for System R

- Principle of optimality allows us to build best subplans “bottom up”
  - Pass 1: Find best plans of height 1 (base table accesses), and record them in a table
  - Pass 2: Find best plans of height 2 (joins of base tables) by combining plans of height 1, record them in a table
  - ... 
  - Pass $i$: Find best plans of height $i$ by combining plans of height $i - 1$ with plans of height 1, record them in a table
  - ... 
  - Pass $n$: Find best plan overall by combining plans of height $n-1$ with plans of height 1.
### The Basic Dynamic Programming Table

Table keyed on 1st column

<table>
<thead>
<tr>
<th>Subset of tables in FROM clause</th>
<th>Best plan</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>{R, S}</td>
<td>hashjoin(R,S)</td>
<td>1000</td>
</tr>
<tr>
<td>{R, T}</td>
<td>mergejoin(R,T)</td>
<td>700</td>
</tr>
</tbody>
</table>
A Note on “Interesting Orders”

• Physical property: Order. When should we care? When is it “interesting”?

• An intermediate result has an “interesting order” if it is sorted by anything we can use later in the query (“downstream” the arrows (operator)):
  – ORDER BY attributes
  – GROUP BY attributes
  – Join attributes of yet-to-be-added joins
    • subsequent merge join might be good
The Dynamic Programming Table

Table keyed on concatenation of 1st two columns

<table>
<thead>
<tr>
<th>Subset of tables in FROM clause</th>
<th>Interesting-order columns</th>
<th>Best plan</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>{R, S}</td>
<td>&lt;none&gt;</td>
<td>hashjoin(R,S)</td>
<td>1000</td>
</tr>
<tr>
<td>{R, S}</td>
<td>&lt;R.a, S.b&gt;</td>
<td>sortmerge(R,S)</td>
<td>1500</td>
</tr>
</tbody>
</table>

Higher cost, but may lead to global optimal plan!
Enumeration of Plans (Contd.)

• First figure out the scans and joins (select-project-join) using dynamic programming
  – **Avoid Cartesian Products** in dynamic programming as follows:
    When matching an \( i-1 \) way subplan with another table, only consider it if
    • There is a join condition between them, or
    • All predicates in WHERE have been “used up” in the \( i-1 \) way subplan.

• Then handle ORDER BY, GROUP BY, aggregates etc. as a post-processing step
  – Via “interestingly ordered” plan if chosen (free!)
  – Or via an additional sort/hash operator

• Despite pruning, this System R dynamic programming algorithm is **exponential** in #tables.
Example

SELECT S.sid, COUNT(*) AS number
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

Pass 1: Best plan(s) for each relation

– Sailors, Reserves: File Scan
– Also B+ tree on Reserves.bid as interesting order
– Also B+ tree on Sailors.sid as interesting order
– Boats: B+ tree on color
### Best plans after pass 1

<table>
<thead>
<tr>
<th>Subset of tables in FROM clause</th>
<th>Interesting-order columns</th>
<th>Best plan</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Sailors}</td>
<td>--</td>
<td>filescan</td>
<td></td>
</tr>
<tr>
<td>{Reserves}</td>
<td>--</td>
<td>Filescan</td>
<td></td>
</tr>
<tr>
<td>{Boats}</td>
<td>--</td>
<td>B-tree on color</td>
<td></td>
</tr>
<tr>
<td>{Reserves}</td>
<td>(bid)</td>
<td>B-tree on bid</td>
<td></td>
</tr>
<tr>
<td>{Sailors}</td>
<td>(sid)</td>
<td>B-tree on sid</td>
<td></td>
</tr>
</tbody>
</table>
Pass 2

// for each left-deep logical plan
for each plan P in pass 1
    for each FROM table T not in P
        // for each physical plan
        for each access method M on T
            for each join method
                generate P ⨝ M(T)
                – File Scan Reserves (outer) with Boats (inner)
                – File Scan Reserves (outer) with Sailors (inner)
                – Reserves Btree on bid (outer) with Boats (inner)
                – Reserves Btree on bid (outer) with Sailors (inner)
                – File Scan Sailors (outer) with Boats (inner)
                – File Scan Sailors (outer) with Reserves (inner)
                – Boats Btree on color with Sailors (inner)
                – Boats Btree on color with Reserves (inner)
        • Retain cheapest plan for each (pair of relations, order)
## Best plans after pass 2

<table>
<thead>
<tr>
<th>Subset of tables in FROM clause</th>
<th>Interesting-order columns</th>
<th>Best plan</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Sailors}</td>
<td>--</td>
<td>filescan</td>
<td></td>
</tr>
<tr>
<td>{Reserves}</td>
<td>--</td>
<td>Filescan</td>
<td></td>
</tr>
<tr>
<td>{Boats}</td>
<td>--</td>
<td>B-tree on color</td>
<td></td>
</tr>
<tr>
<td>{Reserves}</td>
<td>(bid)</td>
<td>B-tree on bid</td>
<td></td>
</tr>
<tr>
<td>{Sailors}</td>
<td>(sid)</td>
<td>B-tree on sid</td>
<td></td>
</tr>
<tr>
<td>{Boats, Reserves}</td>
<td>(B.bid)</td>
<td>SortMerge(B-tree on Boats.color, filescan Reserves)</td>
<td></td>
</tr>
<tr>
<td>Etc...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pass 3 and beyond

• Using **Pass 2 plans** as outer relations, generate plans for the next join in the same way as Pass 2
  – E.g. `{SortMerge(B-tree on Boats.color, filescan Reserves)) (outer) | with Sailors (B-tree sid) (inner)`

• Then, add cost for groupby/aggregate:
  – This is the cost to sort the result by sid, *unless it has already been sorted by a previous operator.*

• Then, choose the cheapest plan
Now you understand the optimizer!

- Benefit #1: You could build one.
- Benefit #2: You can influence one
  - People who write non-trivial SQL often get frustrated with the optimizer
    - It picked a crummy plan!
    - It didn’t use the index I built!
    - Etc.
  - Understanding the optimizer can lead you to:
    - Design your DB & Indexes better
    - Avoid “weak spots” in your optimizer’s implementation
    - Coax your optimizer to do what you want
Summary

- Optimization is the reason for the lasting power of the relational system
- But it is primitive in some SQL databases, and in the Big Data stack
- Many new areas:
  - Smarter statistics (fancy histograms, “sketches”)
  - Auto-tuning statistics
  - Adaptive runtime re-optimization
  - Multi-query optimization
  - Parallel scheduling issues
Reading and Next Class

- Query Optimization: Ch 15
- Next: Security & SQL injection: Ch 21