Selectivity Estimation

Selectivity Estimation

Selectivity Estimation: Equalities

<table>
<thead>
<tr>
<th>PREDICATE</th>
<th>SELECTIVITY</th>
<th>ASSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = value</td>
<td>1 / (# distinct values of A in relation)</td>
<td>We know</td>
</tr>
<tr>
<td>A = value</td>
<td>1 / 10</td>
<td>We don’t know</td>
</tr>
<tr>
<td>A = B</td>
<td>1 / MAX(# distinct A-values, # distinct B-values)</td>
<td>We know</td>
</tr>
<tr>
<td>A = B</td>
<td>1 / (# distinct values of A)</td>
<td>We know</td>
</tr>
<tr>
<td>A = B</td>
<td>1 / 10</td>
<td>We don’t know</td>
</tr>
</tbody>
</table>

|column| = the number of distinct values for the column

If you have an index on column A, you can assume you know |A|, max(A), and min(A)

When using selectivity to compute # of tuples, round up the result (e.g., 245.7 → 246 tuples)
Selectivity Estimation: Inequalities on Integers

<table>
<thead>
<tr>
<th>PREDICATE</th>
<th>SELECTIVITY</th>
<th>ASSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &lt; c</td>
<td>(c – min(A)) / (max(A) – min(A) + 1)</td>
<td>We know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≤ c</td>
<td>(c – min(A) + 1) / (max(A) – min(A) + 1)</td>
<td>c is an integer</td>
</tr>
<tr>
<td>A &lt; c</td>
<td>1 / 3</td>
<td>We don’t know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≥ c</td>
<td>(max(A) – c) / (max(A) – min(A) + 1)</td>
<td>We know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≥ c</td>
<td>1 / 3</td>
<td>We don’t know max(A) and min(A)</td>
</tr>
</tbody>
</table>

* We add 1 to the denominator in order for our [low, high] range to be inclusive
  E.g. range [2, 4] = 2, 3, 4 → (4 - 2) / 1 = 3

Selectivity Estimation: Inequalities on Floats

<table>
<thead>
<tr>
<th>PREDICATE</th>
<th>SELECTIVITY</th>
<th>ASSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &lt; c</td>
<td>(c – min(A)) / (max(A) – min(A))</td>
<td>We know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≤ c</td>
<td>1 / 3</td>
<td>c is a float</td>
</tr>
<tr>
<td>A &lt; c</td>
<td>1 / 3</td>
<td>We don’t know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≥ c</td>
<td>(max(A) – c) / (max(A) – min(A))</td>
<td>We know max(A) and min(A)</td>
</tr>
<tr>
<td>A ≥ c</td>
<td>1 / 3</td>
<td>We don’t know max(A) and min(A)</td>
</tr>
</tbody>
</table>

* We don’t add 1 to the denominator (floats are continuous, integers are discrete)
  E.g. range [2.0, 4.0] = 2.0, 2.1, ..., 3.9, 4.0 → (4.0 - 2.0) = 2.0

Selectivity Estimation: Connectives

<table>
<thead>
<tr>
<th>PREDICATE</th>
<th>SELECTIVITY</th>
<th>ASSUMPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1 AND p2</td>
<td>sel(p1) · sel(p2)</td>
<td>Independent predicates</td>
</tr>
<tr>
<td>p1 OR p2</td>
<td>sel(p1) + sel(p2) – sel(p1) · sel(p2)</td>
<td>Independent predicates</td>
</tr>
<tr>
<td>NOT p</td>
<td>1 – sel(p)</td>
<td></td>
</tr>
</tbody>
</table>

Selectivity Estimation

How many tuples are selected by the following query?

SELECT * FROM R

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]
How many tuples are selected by the following query?

**SELECT * FROM R**

1000 tuples (no predicates, select all)

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

How many tuples are selected by the following query?

**SELECT * FROM R**

WHERE A = 42

50 unique values in A
1/50 · (1000 tuples) = 20 tuples

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]
**Selectivity Estimation**

How many tuples are selected by the following query?

```sql
SELECT * FROM R
WHERE C = 42
```

No information about C

1/10 · (1000 tuples) = 100 tuples

---

**Selectivity Estimation**

How many tuples are selected by the following query?

```sql
SELECT * FROM R
WHERE A <= 25
```

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

\[
\text{sel}(A <= 25) = \frac{(25 - 1 + 1)}{(50 - 1 + 1)} = \frac{1}{2}
\]

1/2 · (1000 tuples) = 500 tuples

---

**Selectivity Estimation**

How many tuples are selected by the following query?

```sql
SELECT * FROM R
WHERE A <= 25
```

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

\[
\text{sel}(A <= 25) = \frac{(25 - 1 + 1)}{(50 - 1 + 1)} = \frac{1}{2}
\]

1/2 · (1000 tuples) = 500 tuples

---

**Selectivity Estimation**

How many tuples are selected by the following query?

```sql
SELECT * FROM R
WHERE B <= 25
```

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

\[
\text{sel}(B <= 25) = \frac{(25 - 1 + 1)}{(50 - 1 + 1)} = \frac{1}{2}
\]

1/2 · (1000 tuples) = 500 tuples
How many tuples are selected by the following query?

\[
\text{SELECT } * \text{ FROM } R \\
\text{WHERE } B \leq 25
\]

\[
\text{sel}(B \leq 25) = (25 - 1) / (100 - 1) = 24/99 = 0.2424...
\]

round(0.2424... · (1000 tuples)) = 242 tuples

---

How many tuples are selected by the following query?

\[
\text{SELECT } * \text{ FROM } R \\
\text{WHERE } C \leq 25
\]

round(1/3 · (1000 tuples)) = 333 tuples

---

How many tuples are selected by the following query?

\[
\text{SELECT } * \text{ FROM } R \\
\text{WHERE } A \leq 25 \\
\text{AND } B \leq 25
\]

---

How many tuples are selected by the following query?

\[
\text{SELECT } * \text{ FROM } R \\
\text{WHERE } C \leq 25
\]

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]
**Selectivity Estimation**

How many tuples are selected by the following query?

\[
\begin{align*}
\text{SELECT} & \quad \* \quad \text{FROM} \quad R \\
\text{WHERE} & \quad A \leq 25 \\
& \quad \text{AND} \quad B \leq 25
\end{align*}
\]

\[
\text{sel}(A \leq 25) \cdot \text{sel}(B \leq 25) = 1/2 \cdot 24/99 = 12/99
\]

\[
\text{round}(0.1212\ldots \cdot (1000 \text{ tuples})) = 121 \text{ tuples}
\]

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

---

**Selectivity Estimation**

How many tuples are selected by the following query?

\[
\begin{align*}
\text{SELECT} & \quad \* \quad \text{FROM} \quad R \\
\text{WHERE} & \quad A \leq 25 \\
& \quad \text{OR} \quad B \leq 25
\end{align*}
\]

\[
\text{sel}(A \leq 25) + \text{sel}(B \leq 25) - \text{sel}(A \leq 25) \cdot \text{sel}(B \leq 25) = 1/2 + 24/99 - 1/2 \cdot 24/99 = 0.62121\ldots
\]

\[
\text{round}(0.62121\ldots \cdot (1000 \text{ tuples})) = 621 \text{ tuples}
\]

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

---

**Selectivity Estimation**

How many tuples are selected by the following query?

\[
\begin{align*}
\text{SELECT} & \quad \* \quad \text{FROM} \quad R \\
\text{WHERE} & \quad A = C
\end{align*}
\]

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]
**Selectivity Estimation**

How many tuples are selected by the following query?

```
SELECT * FROM R
WHERE A = C
```

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

No information about C
1/50 · (1000 tuples) = 20 tuples

**Selectivity Estimation**

How many tuples are selected by the following query?

```
SELECT * FROM R
WHERE A = C
```

R(A,B,C) has 1000 tuples

Attribute A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute B:
100 unique floats, uniformly distributed in the range [1, 100]

No information about C
1/50 · (1000 tuples) = 20 tuples

**Selectivity Estimation**

How many tuples are selected by the following query?

```
SELECT * FROM R, S
WHERE R.A = S.D
```

R(A,B,C) has 1000 tuples

S(D, E) has 500 tuples

Attribute R.A:
50 unique integers, uniformly distributed in the range [1, 50]

Attribute S.D:
25 unique integers, uniformly distributed in the range [1, 25]

Max output size = |R| · |S|

```
sel(R.A = S.D) = 1 / MAX(50, 25) = 1/50
1/50 · (1000 · 500) = 10,000 tuples
```
**Query Optimisation – Background**

We can represent relational algebra expressions as trees.

Order of operators affects I/Os and resource usage, but not necessarily output.

\[ \pi_{sname} (\sigma_{bid=100 \land rating > 5} (Reserves \bowtie Sailors)) \]

**Query Optimisation – Alternate Plans**

Given a plan, some things we can do are:

- Push selections/projections down the tree
- Materialize intermediate relations (write to a temp file)

Materializing intermediate relations results in additional write I/Os, but is better in the long run.

Use indices (e.g., INLJ)

**Query Optimisation – Materialising**

Table R consists of 50 pages
Table S consists of 100 pages

\[ \text{sel}(S.\text{age} < 25) = 0.5 \]

Without materializing, we are performing \( \sigma_{\text{age} < 25} \) on the fly each time in PNLJ, and scanning the entire table S for each page of R.

Cost = Scan R (50) + PNLJ (50 \cdot 100) = 5,050 I/Os
**QUERY OPTIMISATION - MATERIALISING**

Table R consists of 50 pages
Table S consists of 100 pages
sel(S.age < 25) = 0.5

By materializing the intermediate relation, we are applying σ_{age < 25} before PNLJ, and performing the join on the result of the selection

Cost = Scan R (50) + Scan S (100) + Materialise (50) + PNLJ (50 · 50) = 2,700 I/Os

**QUERY OPTIMISATION**

A query optimiser takes in a query plan (e.g., one directly translated from a SQL query) and outputs a better (hopefully optimal) query plan

- Works on and optimizes over a **plan space** (set of all plans considered)
- Performs **cost estimation** on query plans
- Uses a **search algorithm** to search through plan space to find plan with lowest cost estimate
  - May not be optimal (bad estimate or small plan space considered)

**QUERY OPTIMISATION - SYSTEM R**

We will be looking at the System R optimiser (aka Selinger optimiser)

- **Plan space**
  - Only left-deep trees, avoid Cartesian products unless they are the only option
  - **Left-deep trees** represent a plan where all new tables are joined one at a time from the right

- **Cost estimation**
  - Actual System R optimiser incorporates both CPU and I/O cost
  - We will only use I/O cost in this course

- **Search algorithm**
  - Dynamic programming

**QUERY OPTIMISATION - SYSTEM R**

Only consider left-deep plans
**QUERY OPTIMISATION – SYSTEM R**

Why only left-deep trees?
- Join new tables one at a time from the right
- Create an ordering in which to add tables to the query being executed
- Too many possible trees for joins
  - Using only left-deep trees: \( N! \) different ways to order relations
  - Including all permutations tree layouts: A very large number of ways to parenthesize given an ordering (superexponential in \( N \))

<table>
<thead>
<tr>
<th># of relations</th>
<th># of different plans</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>120</td>
</tr>
<tr>
<td>5</td>
<td>1,280</td>
</tr>
<tr>
<td>6</td>
<td>6,720</td>
</tr>
<tr>
<td>7</td>
<td>45,360</td>
</tr>
<tr>
<td>8</td>
<td>403,290</td>
</tr>
<tr>
<td>9</td>
<td>3,628,800</td>
</tr>
<tr>
<td>10</td>
<td>36,288,000</td>
</tr>
</tbody>
</table>

---

**QUERY OPTIMISATION – SYSTEM R**

Search algorithm for System R: use dynamic programming
- Based on the principle of optimality
- Runtime drops from \( n! \) to around \( n \times 2^n \)

To be considered, must be:
- Left deep
- No Cartesian products
  - (I.e. if we join \( R \) and \( S \) on \(<\text{cond1}>\) and we join \( S \) and \( T \) on \(<\text{cond2}>\), we don’t consider joining \( R \) and \( T \) if there’s no condition between them)

---

**QUERY OPTIMISATION – SYSTEM R**

For \( N \) relations joined, perform \( N \) passes
- On the \( i \)-th pass, output only the best plan for joining any \( i \) of the \( N \) relations
- Also keep around plans that have higher cost but have an interesting order

Interesting orders are orderings on intermediate relations that may help reduce the cost of later operators (e.g., joins, sorting, hashing)
- ORDER BY attributes
- GROUP BY attributes
- downstream join attributes

---

**SYSTEM R OPTIMISATION: EXAMPLE**

Pass 1:
- Find minimum cost access method for each (relation, interesting order) pair
  - Index scan, full table scans

A toy example:

```
SELECT * FROM R, S, T
WHERE R.B = S.B
AND S.C = T.C
AND R.A <= 50
```
Pass 1:
Assume the single table access plans have the following IO costs:

- Full scan on R: 1000 I/Os
- Index scan on R.A: 200 I/Os (sorted on R.A)
- Index scan on R.B: 1100 I/Os (sorted on R.B)
- Full scan on S: 2000 I/Os
- Index scan on S.B: 2500 I/Os (sorted on S.B)
- Full scan on T: 3000 I/Os
- Index scan on T.C: 3500 I/Os (sorted on T.C)
- Index scan on T.D: 3500 I/Os (sorted on T.D)

Pass 1:
Which single table access plans advance to the next stage?

- Full scan on R: 1000 I/Os
- Index scan on R.A: 200 I/Os (sorted on R.A)
- Index scan on R.B: 1100 I/Os (sorted on R.B)
- Full scan on S: 2000 I/Os
- Index scan on S.B: 2500 I/Os (sorted on S.B)
- Full scan on T: 3000 I/Os
- Index scan on T.C: 3500 I/Os (sorted on T.C)
- Index scan on T.D: 3500 I/Os (sorted on T.D)

Pass 1:
Which single table access plans advance to the next stage?

- Full scan on R: 1000 I/Os
- Index scan on R.A: 200 I/Os (sorted on R.A)
- Index scan on R.B: 1100 I/Os (sorted on R.B)
- Full scan on S: 2000 I/Os
- Index scan on S.B: 2500 I/Os (sorted on S.B)
- Full scan on T: 3000 I/Os
- Index scan on T.C: 3500 I/Os (sorted on T.C)
- Index scan on T.D: 3500 I/Os (sorted on T.D)
Assume the following join costs:

<table>
<thead>
<tr>
<th>Join</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>R * S</td>
<td>21,000 I/Os</td>
</tr>
<tr>
<td>R * S</td>
<td>3,600 I/Os</td>
</tr>
<tr>
<td>S * R</td>
<td>18,000 I/Os</td>
</tr>
<tr>
<td>S * R</td>
<td>3,000 I/Os</td>
</tr>
<tr>
<td>T * R</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * R</td>
<td>20,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>25,000 I/Os</td>
</tr>
<tr>
<td>S * T</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>S * T</td>
<td>20,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>25,000 I/Os</td>
</tr>
</tbody>
</table>

SELECT * FROM R, S, T
WHERE R.B = S.B
AND S.C = T.C
AND R.A <= 50

Which of these joins will actually be considered by the query optimiser on pass 2?

Select the best join for each considered set of tables.

Assume the following join costs:

<table>
<thead>
<tr>
<th>Join</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>R * S</td>
<td>21,000 I/Os</td>
</tr>
<tr>
<td>R * S</td>
<td>3,600 I/Os</td>
</tr>
<tr>
<td>S * R</td>
<td>18,000 I/Os</td>
</tr>
<tr>
<td>S * R</td>
<td>3,000 I/Os</td>
</tr>
<tr>
<td>T * R</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * R</td>
<td>20,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>25,000 I/Os</td>
</tr>
<tr>
<td>S * T</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>S * T</td>
<td>20,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td>T * S</td>
<td>25,000 I/Os</td>
</tr>
</tbody>
</table>

SELECT * FROM R, S, T
WHERE R.B = S.B
AND S.C = T.C
AND R.A <= 50

Which of these joins will advance to the next pass of the query optimiser?

None of the joins produce an interesting order (no downstream joins, ORDER BY, GROUP BY). Only consider best join for each considered set of tables.
Assume the following join costs:

<table>
<thead>
<tr>
<th></th>
<th>R (\bowtie) BNLJ S</th>
<th>21,000 I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R (\bowtie) SMJ S</td>
<td>3,600 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) BNLJ R</td>
<td>18,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) SMJ R</td>
<td>3,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>R (\bowtie) BNLJ T</td>
<td>30,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>R (\bowtie) SMJ T</td>
<td>40,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) BNLJ R</td>
<td>15,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) SMJ R</td>
<td>10,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>T (\bowtie) BNLJ R</td>
<td>25,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>T (\bowtie) SMJ R</td>
<td>20,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) BNLJ T</td>
<td>15,000 I/Os</td>
</tr>
<tr>
<td></td>
<td>S (\bowtie) SMJ T</td>
<td>10,000 I/Os</td>
</tr>
</tbody>
</table>

Will any of these remaining joins produce an interesting order?

**System R Optimisation: Example**

How could we modify the query so that \(S \bowtie_{SMJ} R\) yields an interesting order?

\(S \bowtie_{SMJ} R\) will be sorted on column B, so we need B to be interesting. We could add ORDER BY B, GROUP BY B, or another join condition involving R.B or S.B to the query to make it interesting.
**SYSTEM R OPTIMISATION: EXAMPLE**

Will the query plan $T \bowtie \text{BNLJ} (S \bowtie \text{SMJ} R)$ be considered by the final pass of the query optimiser?

No, this query plan is **not** left-deep (all join results must be on the left side of their parent join), so it is not considered in the final pass.