THE UNIVERSITY of EDINBURGH

Advanced Database Systems

Spring 2024

Lecture #30:

Revision II

Plan for Today

Selectivity Estimation

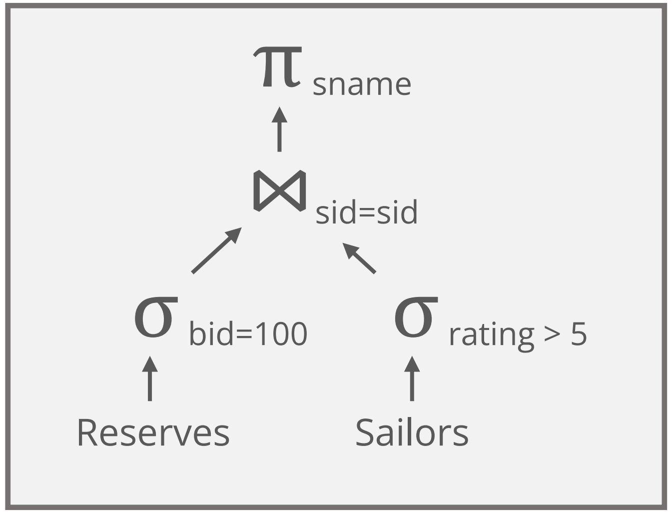
Query Optimisation

Selectivity Estimation

To estimate the cost of a query, we add up the estimated costs of each operator in the query

- Need to know the size of the **intermediate relations** (generated from one operator and passed into another) in order to do this!
- Need to know the **selectivity** of predicates - what % of tuples are selected by a predicate

These are all estimates… if we don't know, we make up a value for it (e.g., selectivity = 1/10)



Selectivity Estimation: Equalities

PREDICATE	SELECTIVITY	ASSUMPTION
A = value	1 / (# distinct values of A in relation)	We know |A|
A = value	1 / 10	We don't know |A|
A = B	1 / MAX (# distinct A-values, # distinct B-values)	We know |A| and |B|
A = B	1 / (# distinct values of A)	We know |A| but not |B|
A = B	1 / 10	We don't know |A| nor |B|

|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

PREDICATE	SELECTIVITY	ASSUMPTION
A < c A ≤ c	(c – min(A)) / (max(A) – min(A) + 1) (c – min(A) + 1) / (max(A) – min(A) + 1)	We know max(A) and min(A) c is an integer
A < c A ≤ c	1 / 3	We don't know max(A) and min(A) c is an integer
A > c A ≥ c	(max(A) – c) / (max(A) – min(A) + 1) (max(A) – c + 1) / (max(A) – min(A) + 1)	We know max(A) and min(A) c is an integer
A > c A ≥ c	1 / 3	We don't know max(A) and min(A) c is an integer

* 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

PREDICATE	SELECTIVITY	ASSUMPTION
A < c A ≤ c	(c – min(A)) / (max(A) – min(A))	We know max(A) and min(A) c is a float
A < c A ≤ c	1 / 3	We don't know max(A) and min(A) c is a float
A > c A ≥ c	(max(A) – c) / (max(A) – min(A))	We know max(A) and min(A) c is a float
A > c A ≥ c	1 / 3	We don't know max(A) and min(A) c is a float

* 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

PREDICATE	SELECTIVITY	ASSUMPTION
p1 **AND** p2	sel(p1) · sel(p2)	Independent predicates
p1 **OR** p2	sel(p1) + sel(p2) – sel(p1) · sel(p2)	Independent predicates
NOT p	1 – sel(p)	

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]

Selectivity Estimation

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]

Selectivity Estimation

How many tuples are selected by the following query?

```
SELECT * FROM R
 WHERE A = 42
```

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 = 42
```

50 unique values in A

$1/50 \cdot (1000 \text{ tuples}) = 20 \text{ tuples}$

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?

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

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 C = 42
```

No information about C

1/10 · (1000 tuples) = 100 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?

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

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 <= 25
```

sel(A <= 25) =

= (25 – 1 + 1) / (50 – 1 + 1)

= 1/2

1/2 · (1000 tuples) = 500 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?

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

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 B <= 25
```

sel(B <= 25) =

= (25 – 1) / (100 – 1)

= 24/99 = 0.2424…

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]

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

Selectivity Estimation

How many tuples are selected by the following query?

```
SELECT * FROM R
 WHERE C <= 25
```

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 C <= 25
```

No information about C

round(1/3 · (1000 tuples)) = 333 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?

```
SELECT * FROM R
 WHERE A <= 25
   AND B <= 25
```

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 <= 25
   AND B <= 25
```

sel(A <= 25) · sel(B <= 25)

= 1/2 · 24/99 = 12/99

= 0.1212…

round(0.1212… · (1000 tuples)) = 121 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?

```
SELECT * FROM R
 WHERE A <= 25
    OR B <= 25
```

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 <= 25
    OR B <= 25
```

sel(A <= 25) + sel(B <= 25)
– sel(A <= 25) · sel(B <= 25)

= 1/2 + 24/99 – 1/2 · 24/99 = 0.62121...

round(0.62121... · (1000 tuples)) = 621 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?

```
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]

Selectivity Estimation

How many tuples are selected by the following query?

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

No information about C

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?

```
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]

Selectivity Estimation

How many tuples are selected by the following query?

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

Max output size = |R| · |S|

sel(R.A = S.D)

= 1 / MAX(50, 25) = 1/50

1/50 · (1000 · 500) = 10,000 tuples

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]

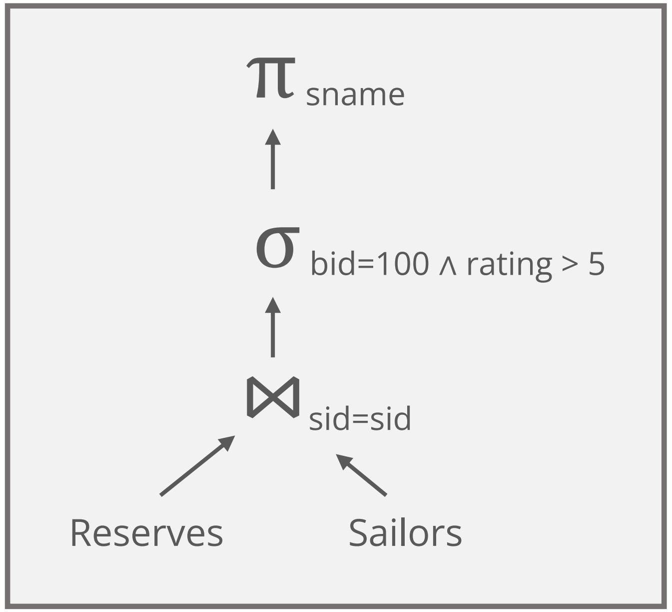
Query Optimisation

Query Optimisation – Background

We can represent relational algebra expressions as trees

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

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



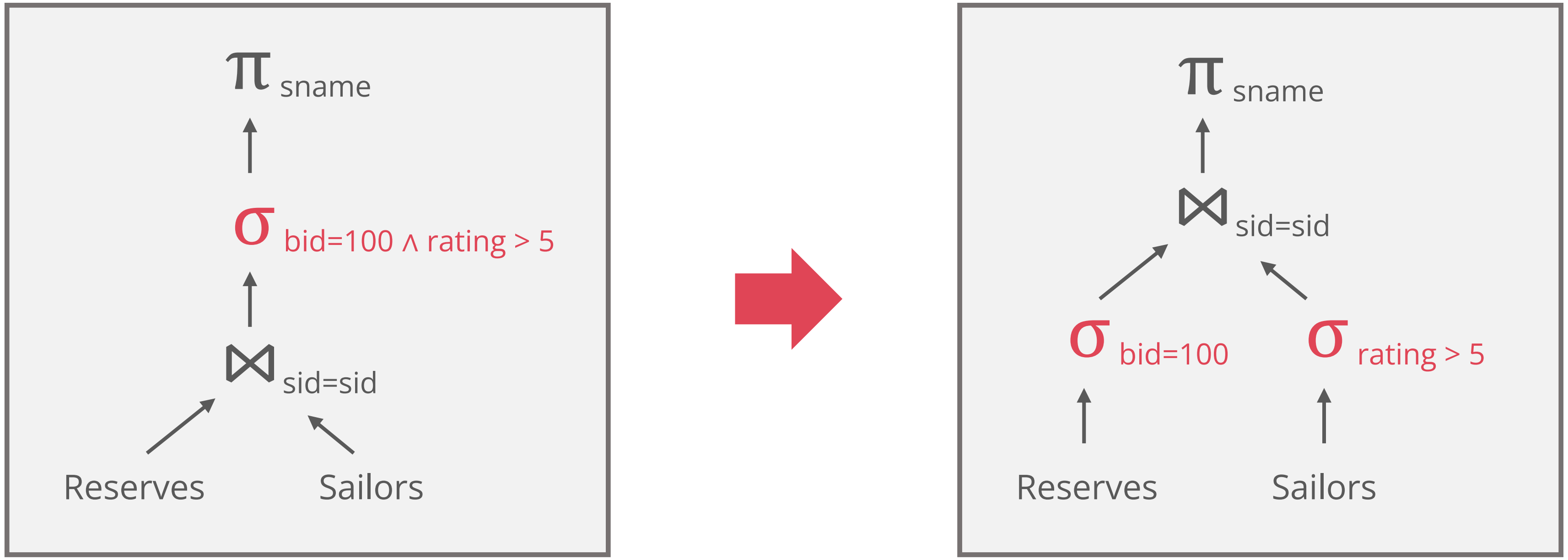
Query Optimisation – Alternate Plans

Given a plan, some things we can do are:

Push selections/projections down the tree

The earlier we reduce the size of input, the fewer I/Os are incurred as we traverse up the tree

Reduces I/O cost if materialized



Query Optimisation – Alternate Plans

Given a plan, some things we can do are:

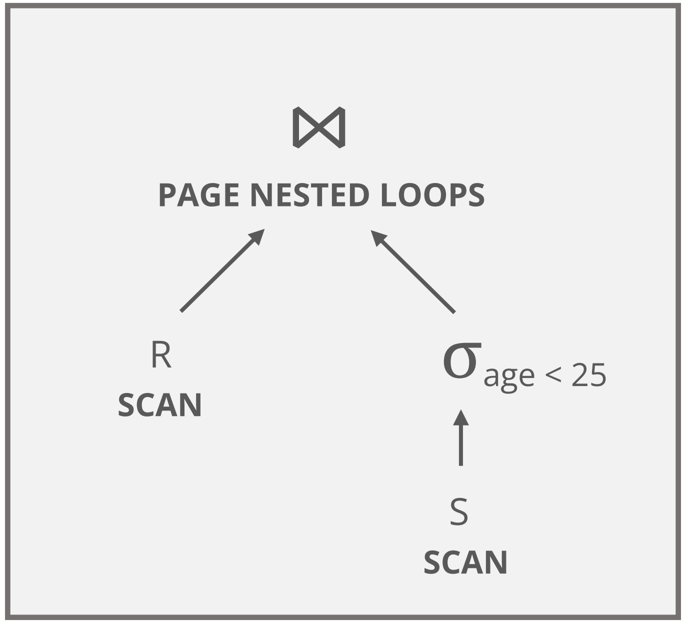
- **Push selections/projections down the tree**
- **Materialize intermediate relations** (write to a temp file)
  - Results in additional write I/Os, but is better in the long run
- **Use indices** (e.g., INLJ)

Query Optimisation - Materialising

Table R consists of 50 pages

Table S consists of 100 pages

sel(S.age < 25) = 0.5



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

Cost = Scan R (50) + PNLJ (50 · 100) = **5,050** I/Os

Query Optimisation - Materialising

Table R consists of 50 pages

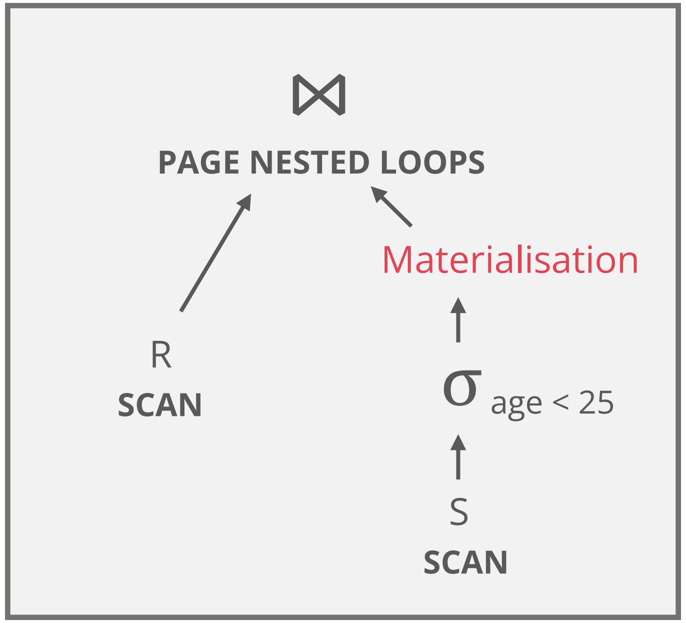
Table S consists of 100 pages

sel(S.age < 25) = 0.5

By materializing the intermediate relation, we are applying $\sigma_{age < 25}$ 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)

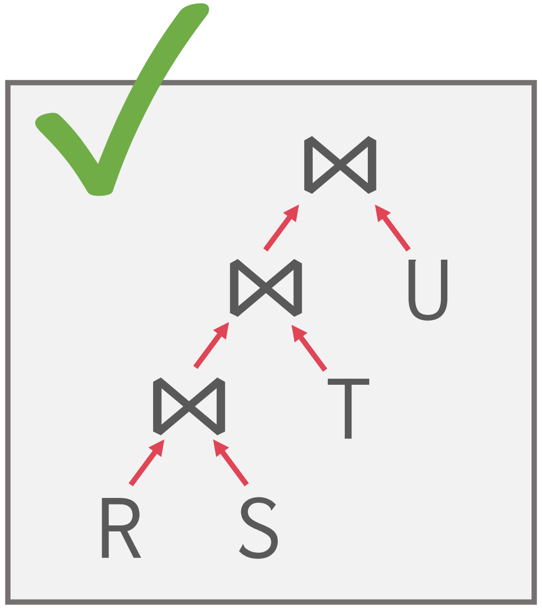
Query Optimisation – System R

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

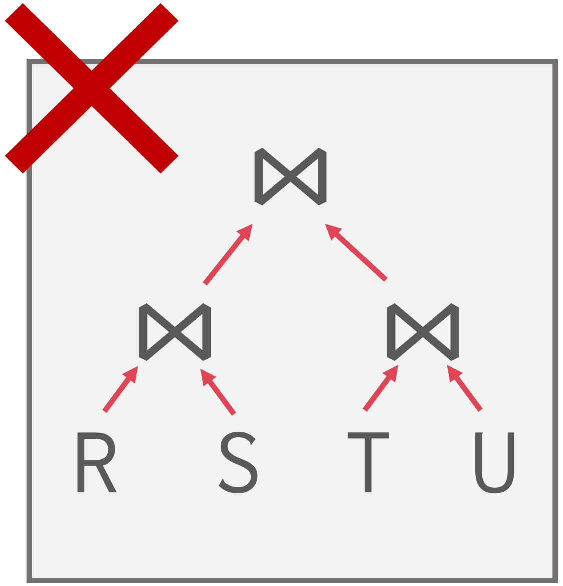
Query Optimisation – System R

Only consider left-deep plans



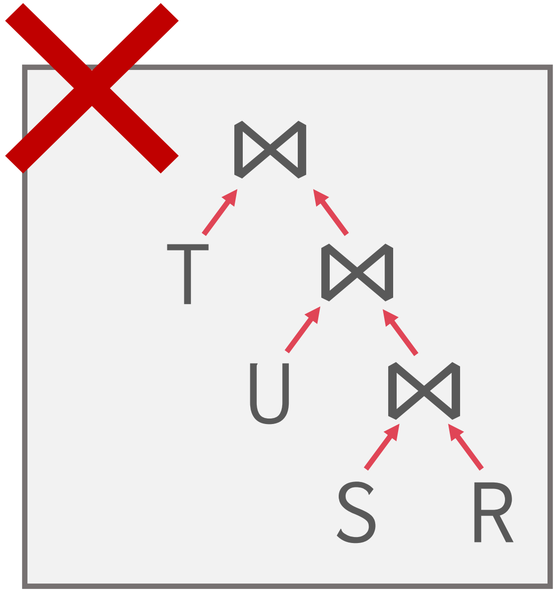
left-deep

((R ⋈ S) ⋈ T) ⋈ U



bushy

(R ⋈ S) ⋈ (T ⋈ U)



right-deep

T ⋈ (U ⋈ (S ⋈ R))

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)

# of relations n	# of different join trees
2	2
3	12
4	120
5	1,680
6	30,240
7	665,280
8	17,297,280
10	17,643,225,600

Query Optimisation – System R

- Search algorithm for System R: use dynamic programming
  - Based on the principle of optimality
  - Runtime drops from **n!** to around **$n \cdot 2^n$**
- To be considered, must be:
  - Left deep
  - No Cartesian products
    - (I.e. if we join R and S on <cond1> and we join S and T on <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
```

System R Optimisation: Example

Pass 1:

Assume the single table access plans have the following IO costs:

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

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)

System R Optimisation: Example

Pass 1:

Which single table access plans advance to the next stage?

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

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)

System R Optimisation: Example

Pass 1:

Which single table access plans advance to the next stage?

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

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)

System R Optimisation: Example

Pass 1:

Which single table access plans advance to the next stage?

- Index scan on R.A (best overall plan for R)
- Index scan on R.B (output sorted on R.B and R.B is used in a join)
- Full scan on S (best overall plan for S)
- Index scan on S.B (output sorted on S.B and S.B is used in a join)
- Full scan on T (best overall plan for T)
- Index scan on T.C (output sorted on T.C and T.C is used in a join)

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	3,000 I/Os

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	10,000 I/Os
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

```
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?

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	3,000 I/Os

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	10,000 I/Os
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

```
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?

Eliminate Cartesian products (joins between R and T)

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	3,000 I/Os

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	10,000 I/Os
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

```
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?

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	**3,000 I/Os**

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	**10,000 I/Os**
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

```
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

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	**3,000 I/Os**

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	**10,000 I/Os**
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

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

Will any of these remaining joins produce an interesting order?

System R Optimisation: Example

Assume the following join costs:

$R \bowtie_{BNLJ} S$	21,000 I/Os
$R \bowtie_{SMJ} S$	3,600 I/Os
$S \bowtie_{BNLJ} R$	18,000 I/Os
$S \bowtie_{SMJ} R$	**3,000 I/Os**

$R \bowtie_{BNLJ} T$	30,000 I/Os
$R \bowtie_{SMJ} T$	40,000 I/Os
$T \bowtie_{BNLJ} R$	35,000 I/Os
$T \bowtie_{SMJ} R$	20,000 I/Os

$S \bowtie_{BNLJ} T$	15,000 I/Os
$S \bowtie_{SMJ} T$	**10,000 I/Os**
$T \bowtie_{BNLJ} S$	25,000 I/Os
$T \bowtie_{SMJ} S$	30,000 I/Os

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

Will any of these remaining joins produce an interesting order?

No. $R \bowtie_{SMJ} S$ is sorted on B (not interesting), and $T \bowtie_{SMJ} S$ is sorted on C (not interesting)

System R Optimisation: Example

How could we modify the query so that S $\bowtie_{SMJ}$ R yields 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 $T \bowtie_{BNLJ} (S \bowtie_{SMJ} R)$ be considered by the final pass of the query optimiser?

System R Optimisation: Example

Will the query plan T $\bowtie_{BNLJ}$ (S $\bowtie_{SMJ}$ 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

Commit
End Transaction