Samarth Kunwar (7014936)
Konstantin Dobrev ()
Felix Ceard-Falkenberg (7007490)

E1

Using

$$C(R \bowtie S) = C(R) + C(S) + C_{\text{HashJoin}}(R \bowtie S) = C(R) + C(S) + |R| + |S|$$

and

$$C(R \times S) = C(R) + C(S) + C_{CP}(R \times S) = C(R) + C(S) + |R| \cdot |S|$$

we can calculate the following table

Subplan	Costs	Result size
Q	0	90
R	0	15
S	0	60
T	0	50
$Q \times R$	1350	1350
$Q \bowtie S$	150	50
$Q \bowtie T$	140	100
$R \bowtie S$	75	40
$R \bowtie T$	65	40
$S \times T$	3000	3000
$(Q \times R) \bowtie T$	2750	400
$(Q \bowtie S) \bowtie T$	1550	56
$(R \bowtie S) \bowtie T$	165	533
$(Q \times R) \bowtie S$	2760	33
$(Q \bowtie T) \bowtie S$	250	56
$(R \bowtie T) \bowtie S$	165	533
$(R \bowtie T) \bowtie Q$	195	400
$(Q \bowtie T) \bowtie R$	255	33
$(R \bowtie S) \bowtie Q$	205	33
$(S \times T) \bowtie Q$	6090	56
$(Q \bowtie S) \bowtie R$	290	33
$(S \times T) \bowtie R$	6015	533
$((Q \times R) \bowtie T) \bowtie S$	3210	10
$((Q \bowtie S) \bowtie T) \bowtie S$	1666	10
$((R \bowtie T) \bowtie Q) \bowtie S$	366	10
$((Q \bowtie T) \bowtie R) \bowtie S$	348	10
$((R \bowtie S) \bowtie T) \bowtie Q$	788	10
$((R \bowtie T) \bowtie S) \bowtie Q$	788	10
$((S \times T) \bowtie R) \bowtie Q$	6638	10
$((Q \times R) \bowtie S) \bowtie T$	2843	10
$((Q \bowtie S) \bowtie R) \bowtie T$	373	10
$((R \bowtie S) \bowtie Q) \bowtie T$	288	10
$((Q \bowtie T) \bowtie S) \bowtie R$	321	10
$((S \times T) \bowtie Q) \bowtie R$	6161	10

Therefore, the optimal order is $((R \bowtie S) \bowtie Q) \bowtie T$

b)

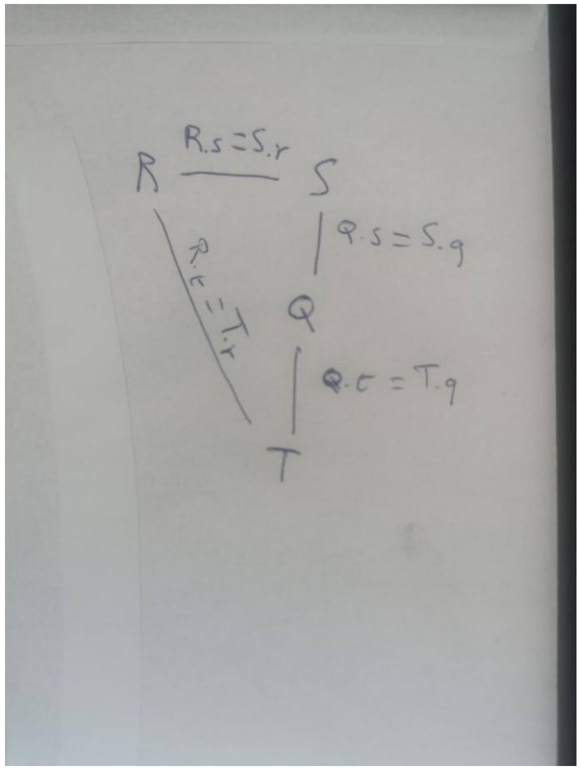


Figure 1: b)

c) Due to the large intermediate results that are included in the cost of joins, plans that involve a Cartesian product must not be considered.

E2

a) i) **Model 1**.

$\sigma_{M.i \geq 182}$.
- scan(M) = 20 000
- index(M, p) = $\log_2(20\ 000) + 57 \cdot 0.01 \cdot 20\ 000 = 11414,287$

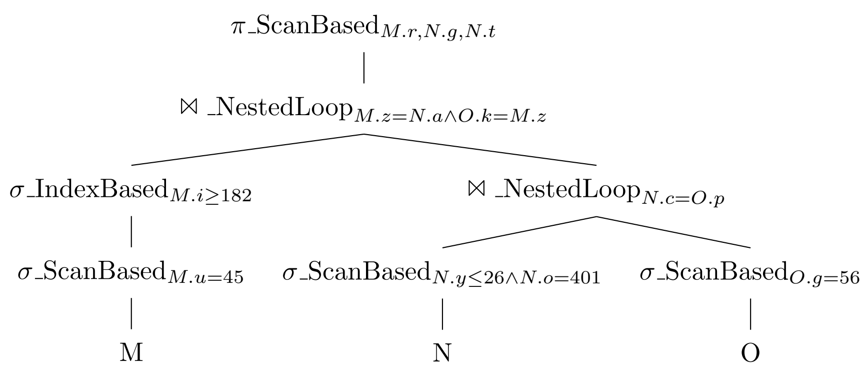
$\sigma_{M.u=45}$.
- scan(M) = 20 000

- index(M, p) $= \log_2(20\ 000) + 57 \cdot 0.3 \cdot 20\ 000 = 342014, 2877$

$\sigma_{O.g=56}$. – scan(M) = 10 000
- index(M, p) $= \log_2(10\ 000) + 57 \cdot 0.1 \cdot 10\ 000 = 57013, 28$

Therefore, for model 1, $\sigma_{M.i\geq 182}$ should use an index access, and $\sigma_{M.u=45}$ and $\sigma_{O.g=56}$ should use a scan.

The updated model tree would look like this:



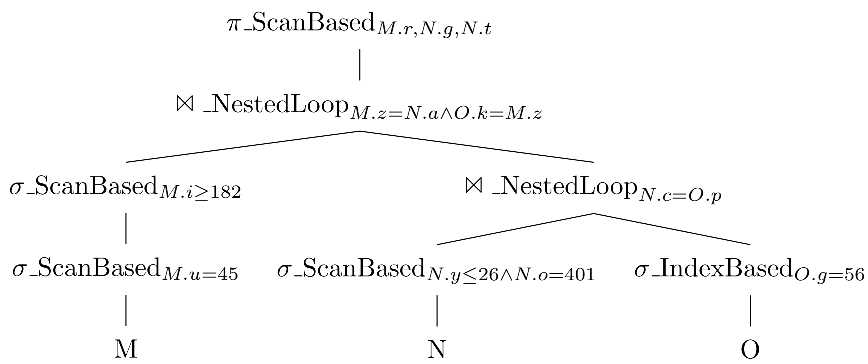
Model 2.

$\sigma_{M.i\geq 182}$. – scan(M) = 50 000
- index(M, p) $= \log_2(50\ 000) + 57 \cdot 0.04 \cdot 50\ 000 = 114015, 609$

$\sigma_{M.u=45}$. – scan(M) = 50 000
- index(M, p) $= \log_2(50\ 000) + 57 \cdot 0.2 \cdot 50\ 000 = 57015, 609$

$\sigma_{O.g=56}$. – scan(M) = 200 000
- index(M, p) $= \log_2(200\ 000) + 57 \cdot 0.015 \cdot 200\ 000 = 171017, 609$

Therefore, for model 2, $\sigma_{M.i\geq 182}$ and $\sigma_{M.u=45}$ should use a scan, and $\sigma_{O.g=56}$ should use an index access.



2.

$$\begin{aligned}
index(T,p) &\leq scan(T) \\
\log_2(|T|) + 57 \cdot sel(p) \cdot |T| &\leq |T| && | - \log_2(|T|) \\
57 \cdot sel(p) \cdot |T| &\leq |T| - \log_2(|T|) && | \cdot \frac{1}{57 \cdot |T|} \\
sel(p) &\leq \frac{|T| - \log_2(|T|)}{57 \cdot |T|} \\
sel(p) &\leq \frac{|T|}{57 \cdot |T|} - \frac{\log_2(|T|)}{57 \cdot |T|} \\
sel(p) &\leq \frac{1}{57} - \frac{\log_2(|T|)}{57 \cdot |T|}
\end{aligned}$$

An index based access is worth Whenever sel(p) is smaller than $\frac{|T|-\log_2(|T|)}{57 \cdot |T|}$. Increasing the size of the table, i.e. T, results in bigger values of $sel(p)$, although it is bounded by $\frac{1}{57}$.

E3

E4

b) If we know that the cost of a subgroup is greater than the smaller total cost, then we could prune the generation of new permutation by removing the one containing these subgroups.

```
// map goes from u.v => count(), min(U.m)
HashMap map;

foreach u in U:
    if u.s == 30:
        if u.v in map.keys():
            if u.m < map[u.v][1]:
                map.update(u.v, (map[u.v][0] + 1, u.m))
             else:
                map.update(u.v, (map[u.v][0]+1, map[u.v][1]))
        else:
            map.insert(u.v, (1, u.m))

data = set()

foreach v in V:
    if v.s >= 30 and v.s < 40:
        if v.u in map.keys():
            data.add([v.u, map[v.u][0], map[v.u][1]])
return data
```

Figure 2: Exercise 3