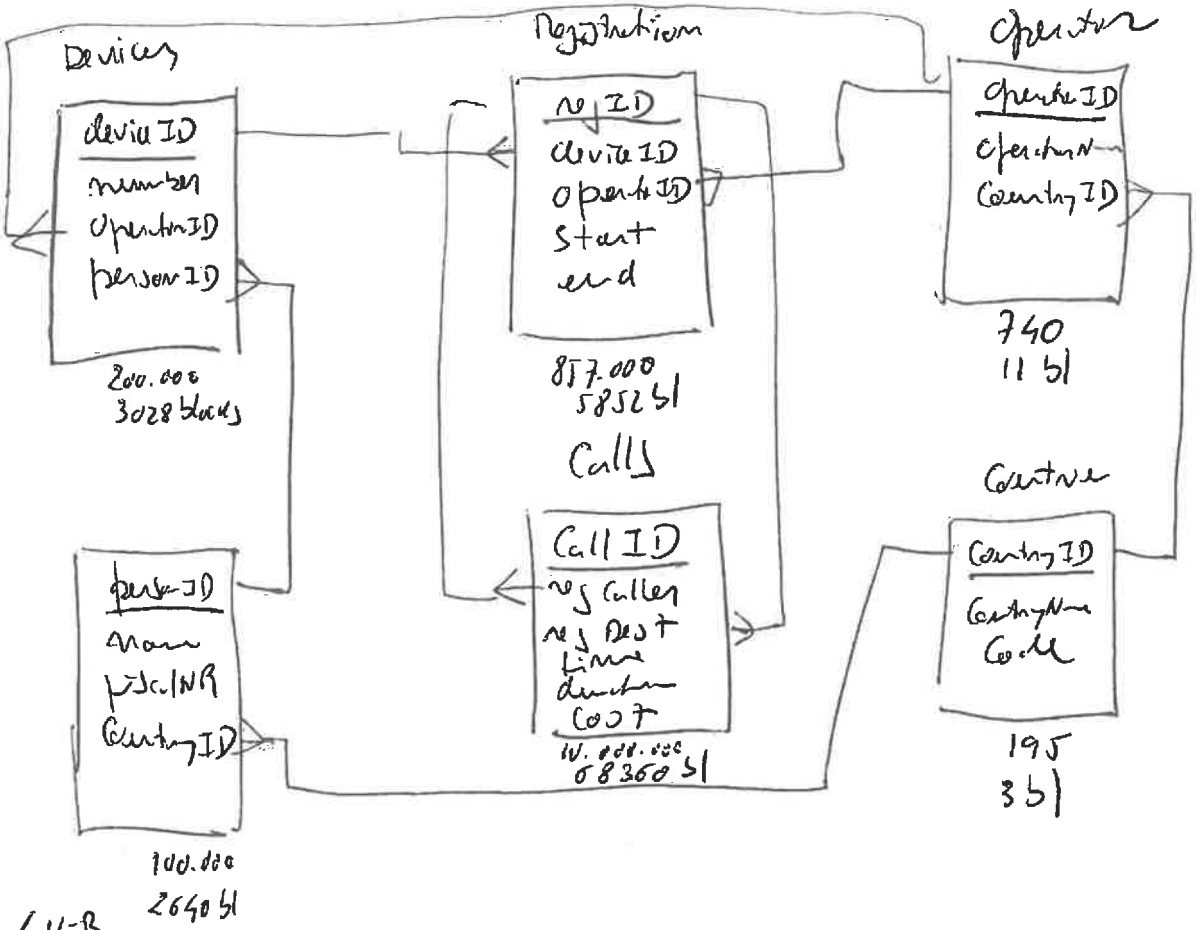
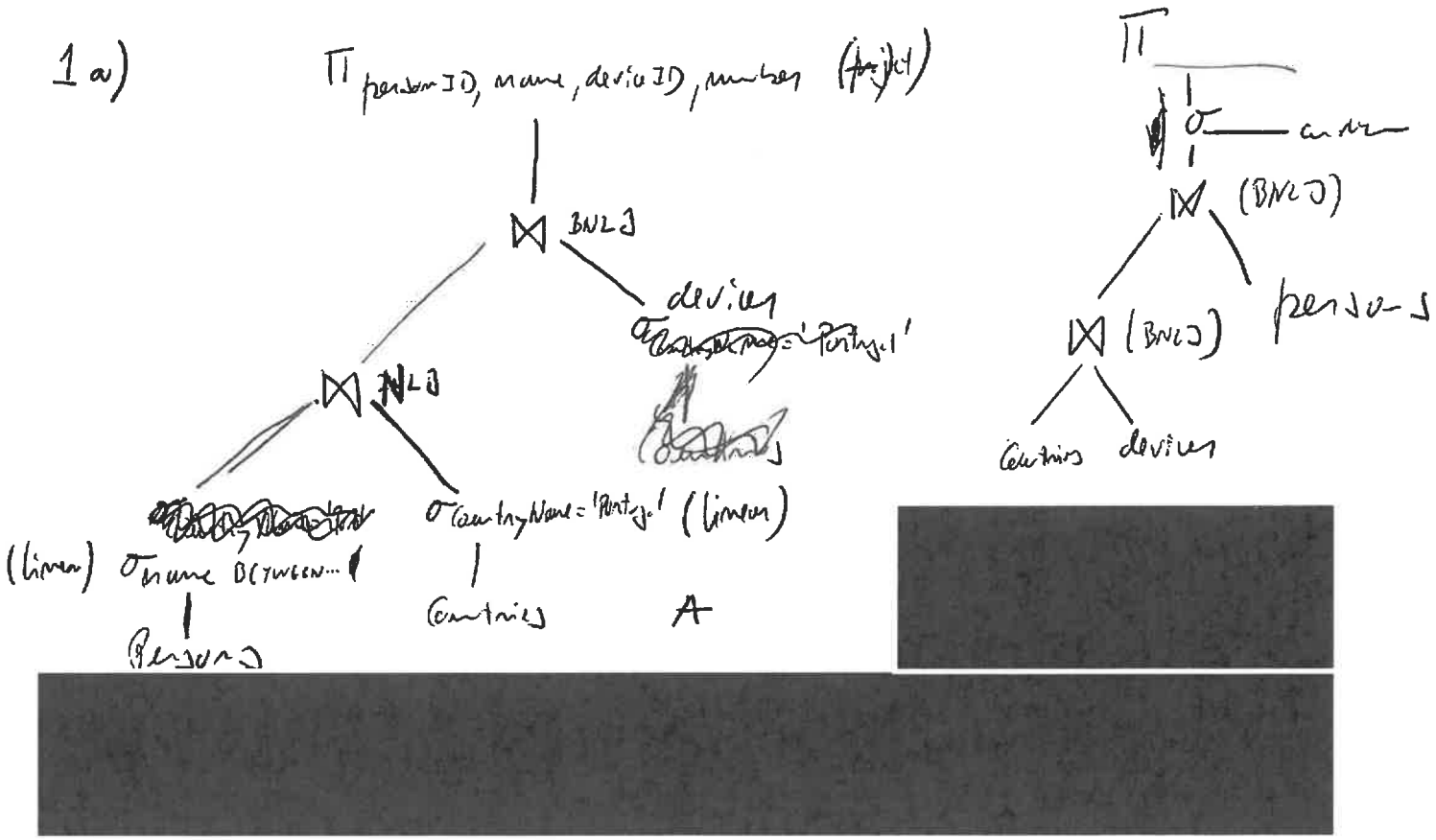


Blocks 4KiB

1 a)

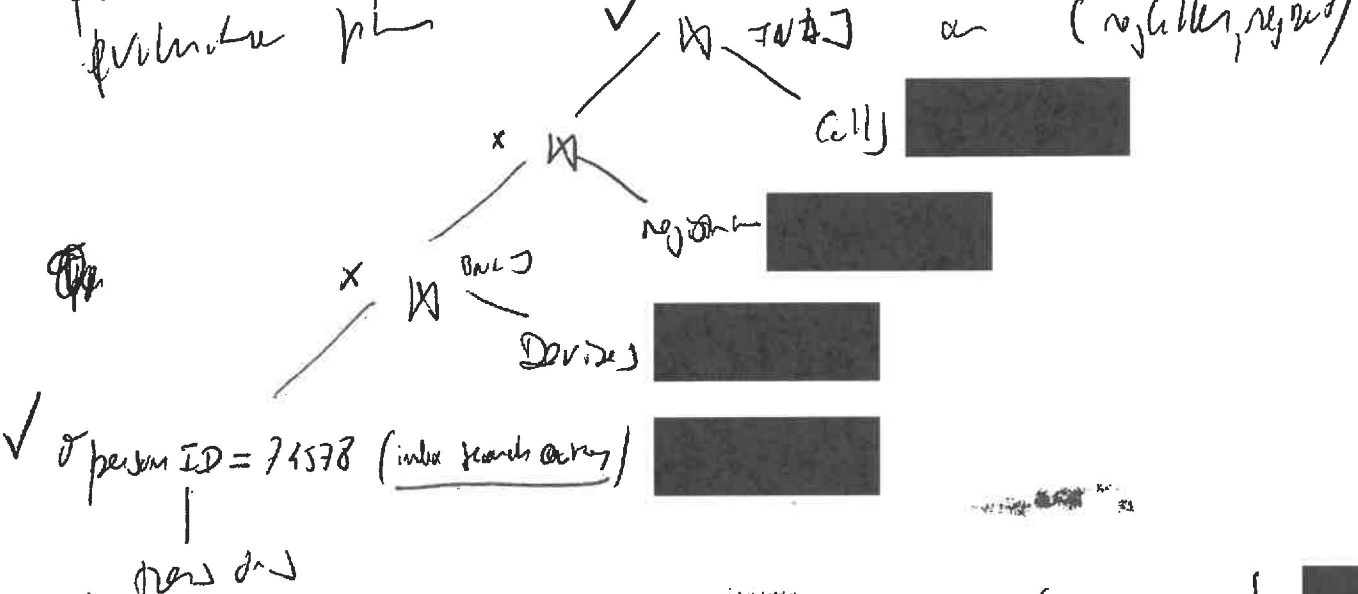


1b)

SELECT *
FROM ~~countries~~ ~~operators~~ operators (NATURAL ~~INNER~~ INNER JOIN countries)
WHERE ~~CountryI~~D = 7
[CountryName = 'Portugal']

[redacted]

1c) ~~The~~ query ~~would~~ in principle could benefit from the following indices in the following evaluation plan



[redacted]

I would create an index on ~~~~ devices(personid), registrations(deviceID), [redacted]

Since the number of expected registrations is 7 devices and 2×500 registrations.

1d) ~~NO~~ [redacted]

SELECT SUM(~~duration~~ cost)
FROM calls ~~WHERE~~
WHERE duration ≤ 1000

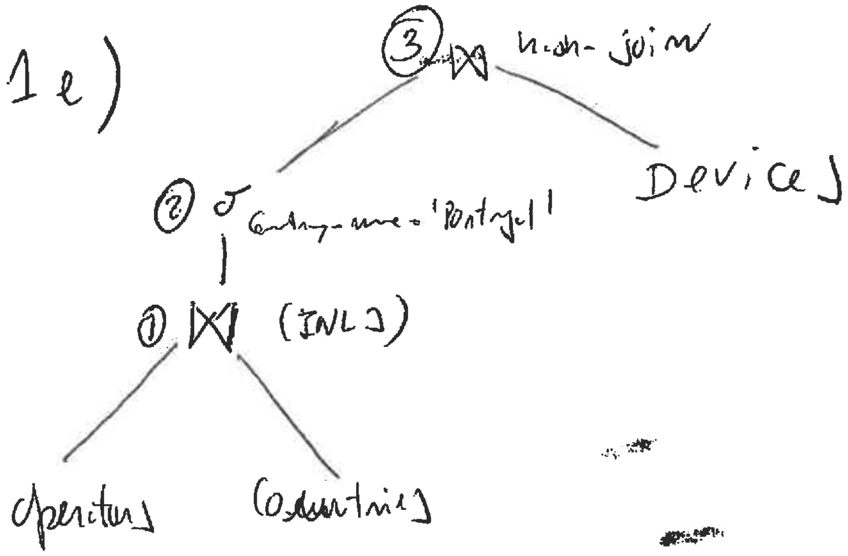
10.000.000 tuples → 305 blocos / [redacted] Vilin [redacted]
teriam de ser consultados 305 000 blocos só para se poder
~~efetuar o cálculo~~! descobrir as tuplos [redacted]
ao contrário [illegible] linear seq [illegible] [illegible] ~~[illegible]~~ [illegible]
[illegible] 68K blocos! [redacted]
[illegible] depois seria necessário [illegible] [illegible] todos!
para [illegible] o custo!

não + pergunta genérica: [redacted]

1d) SELECT SUM(cost)
FROM calls
WHERE duration ≤ 1000

NO: [redacted]
EXP: a lot of values [redacted] + need to get from [redacted]
COST: [redacted]

1e)



① operators is the outer relation with 740 tuples and 11 blocks and the index used will be the primary key of Countries which is searched for so the cost will be

$11*(t_T + t_S) + 740 * c$ [redacted]

so $c = (2+1)*(t_T + t_S)$

therefore the cost is $11*(t_T + t_S) + 740*3*(t_T + t_S)$

$= 11 * 11 t_T + 740*3*11 t_T$ [redacted]

$= 121 t_T + 24420 t_T = 24541 t_T$

(note that the ~~both~~ Operators and the B+-tree and Countries all fit in memory, so one could in fact reduce the time to

$\underbrace{1*t_S + 11*t_T}_{\text{operators}} + \underbrace{1*t_S + 5*t_T}_{\text{B+-tree}} + \underbrace{1*t_S + 3*t_T}_{\text{Countries}}$

B+-tree ↓ could be used to reduce the time?

~~results in 4 tuples~~ results in 740 tuples

② does not cost anything since it can be done simply by filtering in memory

$0*t_S + 0*t_T$ [redacted]

resulting in 4 tuples

③ The hash-join simply constructs a memory hash of the 4 tuples in devicesID of the 4 tuples. Thus the cost is just reading the Devices table (do not need to partition)

$1*t_S + 3028 * t_T$ [redacted] not being in linear order

2a) RAID 6

protects against two disk failures therefore is more fault tolerant. This is important in large disks because of

- latent failures
- errors are more common
- increase rebuild time with one failure given better protection
- reduced cost.

2b) The NULL bitmap is essential to treat correctly Query Conditions like IS NULL/ with respect to additionally it is required to handle correctly the semantics of conditions like NOT(V = 7).

The DELETED BITMAP is used to avoid changing all the bitmaps when a value is deleted (lazy update) this is essential to guarantee that removed values are not returned.

2d)

$1 \times (b_R + b_S)$ → read the relations for partitioning [redacted]

$1 \times (b_R + b_S)$ → write the partitions

$1 \times (b_R + b_S)$ → read the build probe

$(4 \times M_h)$ → for overflow of partitions

$+ 2\left(\frac{b_R}{b_b} + \frac{b_S}{b_b}\right)$ → b_b blocks are read at a time [redacted] → we need to read $\frac{b_R}{b_b}$ blocks for each S → the same for output if we use b_b blocks!

$2 \times M_h$ → seeks → [redacted] read the partitions, one for the probe — one for the build