Solutions of Test-1 and Test-2 of 2022

Companies: 100 x 50 = 5.000 bytes -> 2 blocks
Routes: 2000 x 50 = 100.000 bytes -> 25 blocks
Flights: 200.000 x 25 = 5.000.000 bytes -> 1221 blocks
Travels: 400.000 x 50 = 20.000.000 bytes -> 4883 blocks
VaccinationCert: 800.000 x 100 = 80.000.000 bytes -> 19532 blocks
Persons: 1.000.000 x 200 = 200.000.000 bytes -> 48829 blocks
travelLegs: 20.000.000 x 50 = 1.000.000.000 bytes -> 244.141 blocks

1a)

SELECT * FROM travels INNER JOIN travelLegs USING(code)

Nested loop join with travels as outer relation

1b)

The way the index has been created allows efficient retrieval of travelLeg given a CODE. This is very appropriate to join travels and travelLegs (especially because it is a clustering index) which is expected to be often required. However, it does not allow to retrieve efficiently the information of codes' given the flight which might be necessary for some particular queries (obtain the vaccination certificates for a particular flight)

1c)

I would create a B+-tree index on the search key (NumR,Date)

1d)

SELECT COUNT(*) FROM VaccinationCert WHERE doses = 2 AND Lab='abc'

CREATE BITMAP INDEX idx_doses ON VaccinationCert(doses)
CREATE BITMAP INDEX idx_lab ON VaccinationCert(lab)

This makes sense because both doses and Lab columns have few values.

1e)

SELECT * FROM travels INNER JOIN travelLegs USING(CODE) WHERE code = 12345;

2nd test

1a) SEE THE VIDEO

1b) THE FORMULAE ARE IN THE BOOK

(use smaller relation as outer relation if both do not fit into memory)
$b_r * b_s + b_r$ transfers and $2 * b_r$ seeks

(use smaller relation as inner relation if it fits in memory)

or $b_r + b_s$ transfers and 2 seeks

1b1) travels has 4883 blocks while travelLegs has 244.141 blocks therefore none fits in memory, consequently it is better to use travels as outer relation and travelLegs as inner relation

1b2) the best ordering is ((routes,companies),travelLegs)
The first join can be performed in memory, resulting in at most 2*25 = 50 blocks that afterwards can be used as inner relation to join efficiently with the travelLegs

1c)

SELECT * FROM *travels* **natural inner join** *travelLegs*

1d)

2a)

$\log_{M-1}(br/M) <= 1$ iff br / M <= M-1 iff br <= M*(M-1) aprox. M^2
for instance if we have a memory of 4Gb for 4Kb blocks we can sort in relations that can have around 2^30 * 2^30 blocks = 2^60 blocks = 4^70 Zib …

2b) Consider the join of tables with foreign keys that result in one tuple after each join. By doing a “blind” join minimization it might result in a lot of time trying to optimize the join order.