

Working out depth:

|key| = country\_size + organisation\_size = 4B + 12B = 16B

|value| = 8B (pointer size)

|tuple| = 24B

Node size in bytes = (24 \* tuples\_per\_page) + 8 (extra pointer)

A node is co-designed to fit one per page, so

Node size <= Page Size = 1024B

=> tuples\_per\_page <= (1024 - 8) / 24 = 42.3

We want the most out of the page, so

tuples\_per\_page = 42

|is\_member| = 80000

**Depth of tree = floor(log\_42 (80000)) = 3**

Working out the number of pages:

Min number of leaves when each leaf is as full as possible:

|is\_member| = 80000

Depth 3 (Leaves) : ceil(80000/42) = 1905 pages

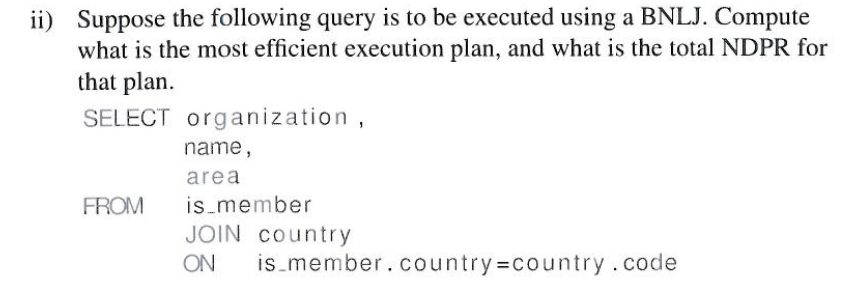
Depth 2 : ceil(1905/42) = 46 pages

Depth 1 : ceil(46/42) = 2 pages

Depth 0 : ceil(2/43) = 1 page (sanity check - why is it 1 page?)

**#pages = 1905 + 46 + 2 + 1 = 1954**

**2aii, 2aiii not in 2019/20 onwards specification**



Pushing projections past the join, we can reduce the required number of pages to load.

Scanning in, we have:

RowSize(country) = 32 + 4 + 35 + 9 + 4 = 84 => 12 tuples/page

RowSize(is\_member) = 4 + 12 + 30 = 46 => 22 tuples/page

Country : ceil(195/12) = 17 pages, DPR

Is\_member : ceil(80000/22) = 3637 pages, DPR

Assuming ~~bulk~~ volcano processing, we get the projections’ intermediate pages for free:

RowSize(country\_proj) = 32 + 4 + 9 = 45 => 22 tuples/page

RowSize(is\_member\_proj) = 4 + 12 = 16 => 64 tuples/page

Country\_proj : 9 pages

Is\_member\_proj : 1250 pages

BNLJ

Joining cost, assuming that from where we start the join, all pages must be reloaded:

Country can be held as the inner tuple in our buffer pool with LRU, since 9 < 20 so the cost is just 9 + 1250 = 1259 DPR**.**

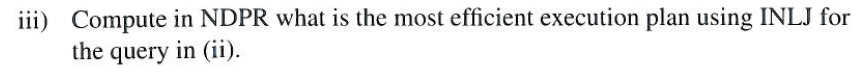
Final Projection

We have another projection to do because we kept is\_member.country as a key to join with, but we don’t actually want it in the end result. We also still need to load the result from the join, which was too large to fit into our buffer.

|joined\_tuples| = |is\_member\_proj| = |is\_member| = 80000

RowSize(joined\_tuples) = 45 + 16 = 61 => 16/page => 5000 DPR

**Total cost: 5000 + 1259 + 3637 + 17 = 9913 DPR**



Using a B+ tree, we can improve our result by indexing on is\_member(country, organisation).

Indexing I/O, per country tuple:

* ~~Assume root is always kept in memory, no faults~~
* ~~Assume 1 fault per non-root~~  Assume 1 fault per node
* Assume tuples to join with are all on the same page

Faults per tuple = 4

Is\_member : 195 \* 4 = 780 DPR

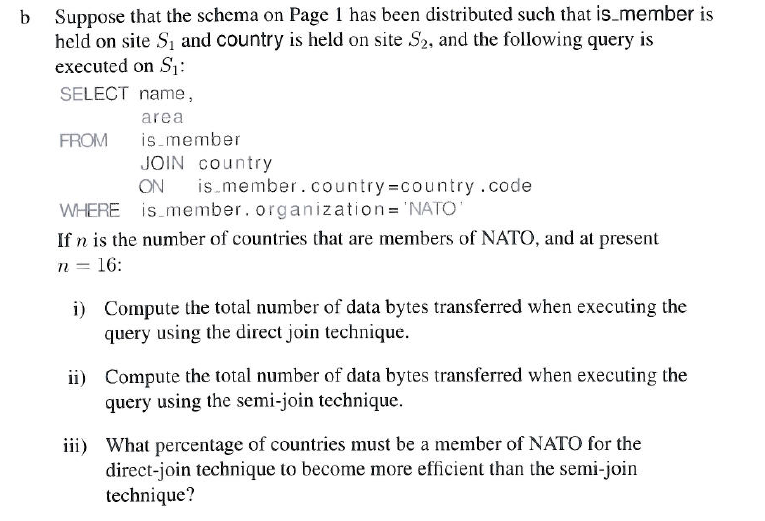
Scan I/O:

* Assume no scan eviction between tuples as indexing uses only 4 pages, 4 + 1 < 20

Country : 9 DPR

INLJ cost: 789 DPR

**Total cost: 5000 + 789 + 3637 + 17 = 9443 DPR**



b)

i)

(32 + 9 + 4) x |country| = 45 |country| = 8775 bytes

ii) s1->s2 = 4n

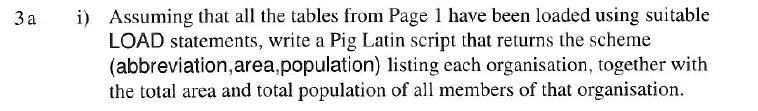
s2->s1 = 45n

Total = 4n + 45n = 49n = 49 x 16 = 784 bytes

iii) Equating (i) with (ii):

( (45 |country| ) / 49n ) = 1, so n = 179.08..

Percentage of countries is 100% x ( n / |country| ) = 100% x ( 179.08.. / 195) = 91.8%

Org\_members = FILTER is\_member BY type == ”member”;

Country\_org\_join = JOIN org\_members BY country,

country BY code;

Country\_org\_data = FOREACH country\_org\_join

GENERATE (is\_member.organisation, code, area, population);

Country\_org\_group = GROUP country\_org\_data

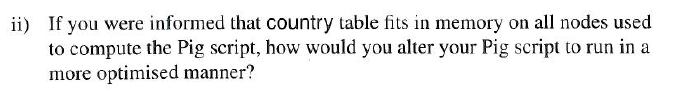
BY organisation;

organisation\_totals = FOREACH country\_org\_group

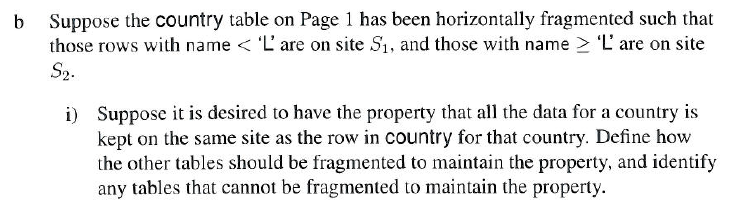
GENERATE (group AS abbreviation, country\_org\_group.code,

SUM(country\_org\_data.area) as area,

SUM(country\_org\_data.population) as population);



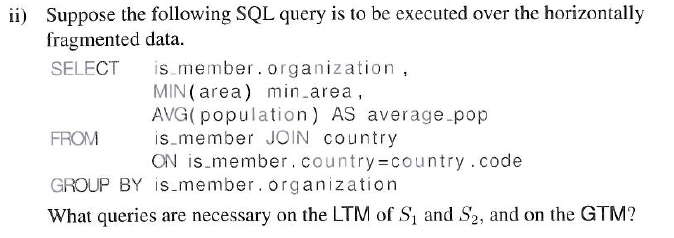
Join REPLICATED (copy the country table to each map node since it fits)



For each table apart from borders we can have (for i = 1, 2):

table\_i = table ⋉ country\_i

Borders cannot be fragmented, as a choice of country1 or country2 could leave the other corresponding to an entry on a different site. (e.g. CZ borders both A, PL)



LTM1:

BEGIN TRANSACTION LT\_1

SELECT is\_member.organisation,

MIN(area) AS min\_area,

SUM(population) as sum\_pop,

COUNT(population) as num\_pop

INTO Itm1

FROM is\_member JOIN country

ON is\_member.country = country.code

GROUP BY is\_member.organisation

COMMIT TRANSACTION LT\_1

LTM2:

BEGIN TRANSACTION LT\_2

SELECT is\_member.organisation,

MIN(area) AS min\_area,

SUM(population) as sum\_pop,

COUNT(population) as num\_pop

INTO Itm2

FROM is\_member JOIN country

ON is\_member.country = country.code

GROUP BY is\_member.organisation

COMMIT TRANSACTION LT\_2

GTM:

SELECT organisation,

MIN(area) AS min\_area,

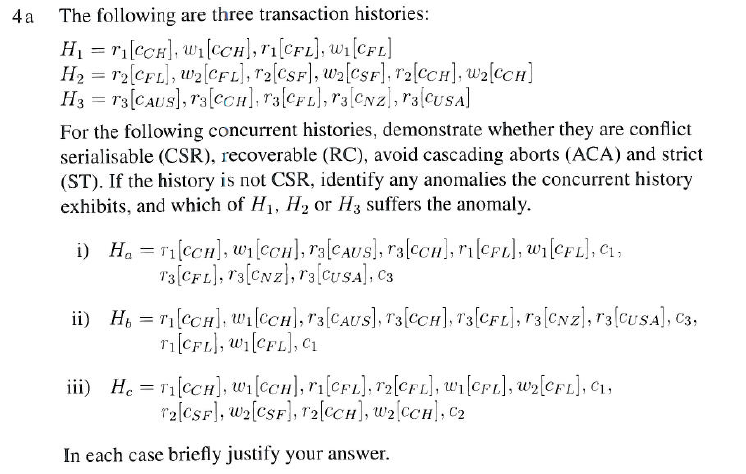
(SUM(sum\_pop) / SUM(num\_pop)) AS average\_pop

FROM Itm1 UNION Itm2

GROUP BY organisation

-------------------------------------------------------------------------------------------------------------------------------

Alternative: send the query without grouping to both sites, then union ltm1 and ltm2 and group in GTM and apply aggregates



a i)

CSR, RC but not ACA as dirty read.

ii)

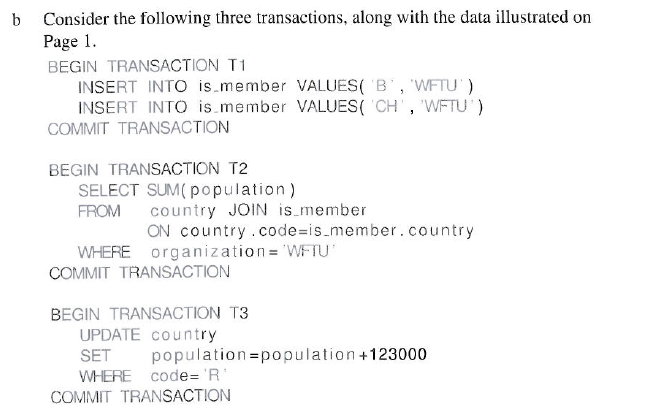
not CSR: w1[CH] < r3[CH], r3[FL] < w1[FL],

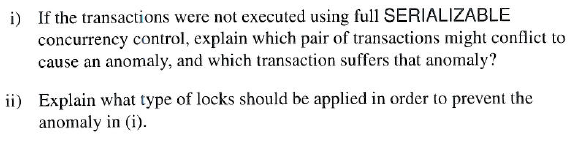
not RC: H3 commits using w1[CH] before H1 has committed.

iii)

not CSR: write skew before committing (r2[FL] < w1[FL], r1[FL] < w2[FL])

Is ACA as no dirty reads.





i)

If T2 is executed in between T1’s insertions, then inconsistent analysis will occur.

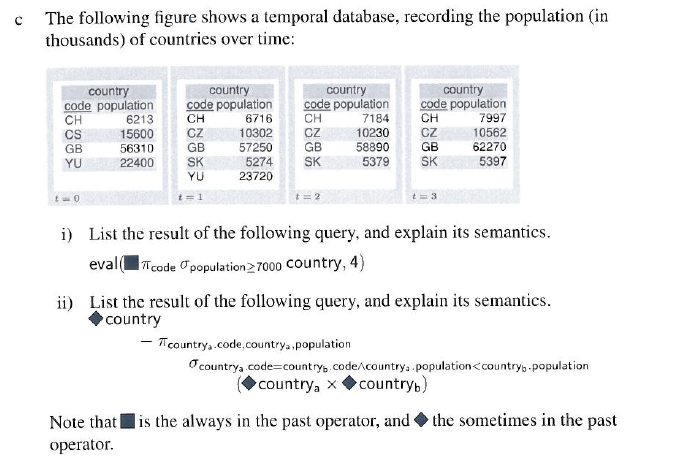
Updates from T3 could also affect the result of T2, again resulting in inconsistent analysis.

* SERIALIZABLE prevents every anomaly except write skew and write skew can happen due to T1 executing in the middle of execution of T2, leading T2 to ignore two entries organisation WFTU in calculation of the sum of population.

ii)

2PL

* Write skew can be prevent by 2PL with predicate locking.



c i)

Find countries which have always existed with population greater or equal to 7000:

{ GB }

ii)

Find the entries with maximum population for each country:

{

CH, 7997,

CS, 15600,

CZ, 10562,

GB, 62270,

SK, 5397,

YU, 23720

}