**QUESTION 2**

**ai)**

| uses | = 120000

Key = (rid, name, opus) => | key | = 4 + 12 + 8 = 24B

Value = ptr to tuple => | value | = 4B (32 bit addressable)

=> | tuple | = 24 + 4 = 28B

Num\_tuples\_per\_page = floor ( (pg\_size - extra\_ptr\_size) / tuple\_size )

= floor ( (1024 - 4) / 28 ) = 36

Pages are unspanned so floor was taken

Depth = floor( log\_36 (120000) ) = 3

Number of pages:

Depth 3 (leaves): ceil( 120000 / 36) = 3334

Depth 2: ceil (3334 / 36) = 93

Depth 1: ceil ( 93 / 36 ) = 3

Depth 0 (root): ceil ( 3 / 36 ) = 1

Total pages: 3334 + 93 + 3 + 1 = 3431

**aii)**

Pushing projections through join gives the following steps:

1. Apply filter to recording
2. Project recording and uses
3. Perform join
4. Project result

1.

(IN)

| recording | = 100000

RowSize(recording) = 4 + 4 + 4 = 12B

TotalSize = 100000 \* 12 = 1200000 => ceil ( 1200000 / 1024 ) = 1172 pages

But because we’re using unspanned pages:

pg size = 1024B

tuples/page \* 12 <= 1024

tuples/page = 85

celi(100,000/85) = 1177 pages

(OUT)

TotalSize = 1/50 \* 100000 \* 12 = 24000 => 24 pages (fits in buffer so no I/O if we keep it there)

2.

Apply projection to recording whilst it’s in buffer so no I/O cost for read or write

(fwiw recording\_proj occupies 16 pages)

(IN)

|uses| = 120000

RowSize(uses) = 4 + 12 + 8 + 4 = 28

TotalSize = 120000 \* 28 = 3360000B => 3282 pages

(OUT)

RowSize(uses\_proj) = 4 + 8 + 4 = 16

TotalSize = 120000 \* 16 = 1920000 = 1875 pages

3.

(IN)

recording\_proj is in buffer already so no I/O

Uses\_proj is read once => 1875 pages

(OUT)

| join | = | recording\_proj | (assume) = 2000

RowSize(join) = 12 + 16 = 28

TotalSize = 56000 => 55 pages

4.

(IN)

55 pages

(OUT)

2000 \* (4 + 8 + 4 + 4) = 40000 => 40 pages

TOTAL: 1. + 2. + 3. + 4.

= (1172) + (3282 + 1875) + (1875 + 55) + (55 + 40) = 8354 NDPR

**aiii)**

Use index from 2ai

Depth = 3 so each tuple access takes 4 page faults

If filter is applied first we get the same cost from 1. in 2aii (1172) and 2000 tuples are left in buffer

Look up on 2000 tuples costs 2000 \* 4 = 8000 and result is written to 55 pages

As before the result must be projected so cost from 4. in 2aii (95)

TOTAL = 1172 + 8000 + 55 + 95 = 9322

**bi)**

Anomaly: inconsistent analysis

If T2 executes between T1’s two updates it will have an inconsistent view of the data

Since T1 names specific objects, use read/write locks

Non-recoverable if T1 aborts

**bii)**

Anomaly: dirty read

If T3 acts on T2’s insert before it commits

Use strict read/write locks

Non-recoverable it T3 commit before T2

**QUESTION 3**

**a)**

Composes\_bag =

GROUP composes

BY opus;

Composes\_counted =

FOREACH composes\_bag

GENERATE name, opus, COUNT(opus) as count;

Composes\_filtered =

FILTER composes\_counted

BY count = 1;

Composes\_proj =

FOREACH composes\_filtered

GENERATE name, opus;

Compose =

JOIN composer by name,

Composes\_proj by name;

Music\_proj =

FOREACH music

GENERATE opus, title;

Result =

JOIN compose by opus,

Music\_proj BY opus;

**bi)**

The join is on the same attribute as the fragmentation so send the query to LTM1 and LTM 2

GTM:  
SELECT \*

FROM (LTM1 UNION ALL LTM2)

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

Alternative bi) Basically copied from a similar question from 2013-2014 paper, correct me if im wrong

LTM1:

BEGIN TRANSACTION LT\_1

SELECT recording.made,

COALESCE(SUM(uses.royalty), 0.0) AS total\_royalty,

COUNT(uses.royalty) AS count\_royalty

INTO Itm1

FROM recording LEFT JOIN uses

ON recording.rid = uses.rid

GROUP BY recording.made

COMMIT TRANSACTION LT\_1

LTM2:

BEGIN TRANSACTION LT\_2

SELECT recording.made,

COALESCE(SUM(uses.royalty), 0.0) AS total\_royalty,

COUNT(uses.royalty) AS count\_royalty

INTO Itm2

FROM recording LEFT JOIN uses

ON recording.rid = uses.rid

GROUP BY recording.made

COMMIT TRANSACTION LT\_2

GTM:

SELECT recording.made,

COALESCE(SUM(total\_royalty), 0.0) AS total\_royalty,

(SUM(total\_royalty) / SUM(count\_royalty)) AS avg\_royalty

FROM Itm1 UNION Itm2

GROUP BY recording.made

**bii)**

Name appears in 3/10 relations as the only key so use it as the attr for fragmentation

artist1 = select\_{name < ‘L’} (artist)

artist2 = select\_{name >= ‘L’} (artist)

composer\_i = artist\_i ⋉ composer

musician\_i = artist\_i ⋉ musician

**biii)**

List the foreign keys from the schema which do not have name as a single key

= { opus, pid, rid }

**QUESTION 4**

**ai)**

After projection pushed through join:

RowSize(recording) = 8

RowSize(uses) = 8

| recording | = 100000

| uses | = 120000

S2 -> S1 = 120000 \* 8 = 960000

**aii)**

S1 -> S2 = 100000 \* 4 = 400000

S2 -> S1 = 120000 \* 8 = 960000

Total = 1360000

**aiii)**

Hash join

`made` isn’t used in the join????

**bi)**

The last time a recording had at least 1000 sales in a year

rid | format | time

400 | MP3 | 3

222 | CD | 0

**bii)**

Records which have sales since it had uses

88, 400, 222, 304

**biii)**

Records which have always sold more copies on CD than MP3s

222, 31, 88

**biv)**

PROJ\_rid,format,time ( SELECT\_qty==max (

sales X time

NATJOIN

PROJ\_max(qty) AS max (GROUPBY\_rid,format,time (sales X time) )