Q1.

A.

Business (bid CHAR(10), btype VARCHAR(30), bname VARCHAR(60), bcity VARCHAR(30)

,baddr VARCHAR(60), bprovince CHAR(2), bstartdate DATE)

Bytes required per one **business** tuple: 10+30*2/3+60*2/3+30*2/3+60*2/3+2+10= **142** bytes

Users (uemail VARCHAR(45), uname VARCHAR(45), udateofbirth DATE, ujoindate DATE,

uprovince CHAR(2))

Bytes required per one **user** tuple: 45*2/3+45*2/3+10+10+2= **82** bytes

Reviews (reviewid INT, bid CHAR(10), uemail VARCHAR(45), rstars SHORT, rtext

VARCHAR(600), rdate DATE)

Bytes required per one **review** tuple: 8+10+45*2/3+1+600*2/3+10= **459** bytes

Bytes required for buisness table: 142*20,000=2,840,000 bytes
Bytes required for user table: 82*90,000=7,380,000 bytes
Bytes required for review table: 459*90,000*20=826,200,000 bytes

For data page: each page: 4,000 bytes; fill factor 75%; filled bytes: 4,000*75%=3,000 bytes

Pages required for **buisness**: 2,840,000/3,000 = **947** pages
Pages required for **users**: 7,380,000/3,000 = **2,460** pages
Pages required for **reviews**: 826,200,000/3,000 = **275,400** pages

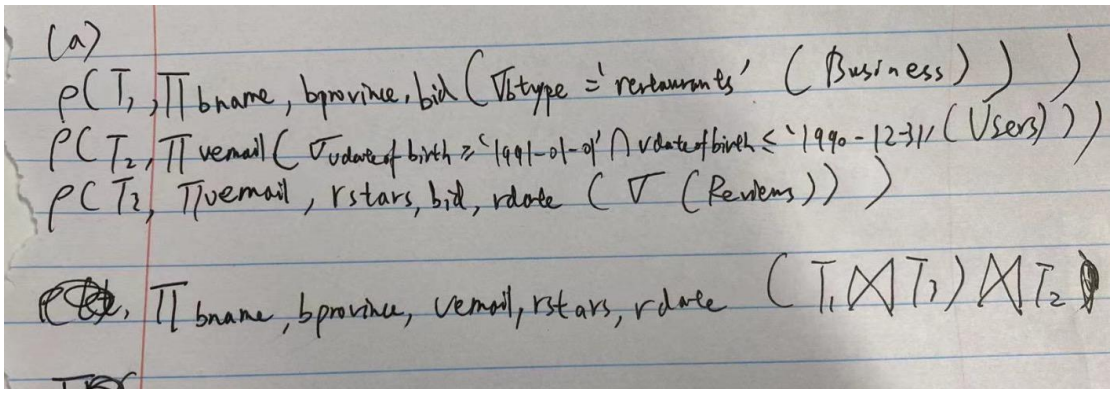
B.

1.Unclustered type1:
Data page:
(20*90,000/20,000) = 90 reviews average per business
For each page 3000/459= 7 tuples per page
90 / 7 = 13 pages data total review
If N = 1, there is 100% reviews; N = 2, 90% reviews; N =3, 65% reviews; N=4, 40% reviews; N=90% reviews.
Suppose the percentage is X, there should be **X*13** data pages for cost of data pages since it's unclustered and the data are scattered. The **worst case** would be all **13** leaf data pages are read.
index leaf page:
Read all indexes where bid = B in the index leaf page, there are 90 indexes.
Structure for index: (bid,rid 1),(bid, rid 2)
Bytes for one index: 10 (bid) + 10 (rid) = 20 bytes

rids capacity per leaf page = 4000*75%/ (20) = 150 indexes per page
There are 90 indexes, so **one** leaf page could give us all the indexes. We only need to read one leaf page and at most **two** when the indexes are in two consecutive pages.
Total IO: 2 leaf pages + 13 data pages
2.Clustered:
For index leaf page:
Index structure (bid, (rid 1, rid 2, rid 3, rid 4,...))
Size: 10 + 90*10 =910 bytes
Indexes per page: 4000*0.75/910 = 3 indexes
We only read one index for the bid. So cost is **one** leaf page.
Data page:
Like in q1, X for percentage of reviews, X*90 would be number of reviews.
One data page contains 7 reviews.
We need to read **X*90/7** data pages.
Total IO: 1 leaf page + X*90/7 data pages

3. Multi-attribute
There are 90 reviews as we know from q1.
It's multi-attribute type2, that means it should be like (bid,rstars,(rid1,rid2,rid3,...))
There are from 1 to 5 rstars so we have 5 indexes per business.
The size of one index would be 10(bid) + 1(rstars) + 90/5(average reviews of one rstar)*10(rid) = 191 bytes
One leaf page can hold 4000*75%/191= 15 indexes
Here we only need to read one business, we will read 5 indexes at most
and thus **one** leaf page can contain 5 indexes, or at most 2 when they are not in one page.
For data pages,
Since it's unclustered, we basically still read X*13 data pages like in q1. And in worst case scenario, we need to read all 13 data pages.
Total IO: 1 leaf page + 13 data pages

Ex.3
A.



B. Do what I did in A
First I project three tables with only attributes I need and which match conditions to save a lot of

pages.

For Business, 20,000/100 = 200 tuples ; bytes for a tuple =bname+bprovince+bid = 60*2/3+2+10=52bytes; total bytes=52*200=10,400 bytes; data pages: 10,400/4000 = **3** pages

For User 1825 tuples born in 1990; bytes for a tuple =uemail=45*2/3=30 bytes; total bytes=30*1825=54,750 bytes; data pages: 54,750/4000 = **14** pages

For Review 18,000,000 tuples; bytes for a tuple =uemail+rstars+rdate+rid=45*2/3+1+10+10=51 bytes; total bytes=51*18,000,000=91,800,000 bytes; data pages: 91,800,000/4000 = **22,950** pages; Have to use this to disk.

Total IO for now is 3+14+22,950=22,967 IO

Join B and R first

For Block nested loop join: cost is 22,950/100*3 = 689 IO when business is outer page and review is inner.

For index nested loop join, we would utilize bid index.

Total leaf page would be: 200 tuple * (10+10)bytes per index / 3000 = 2 pages

The cost to find a B would be: 3 data pages (to read all data pages) + 2 leaf pages

Cost of index nested loop join is 18,000,000(cardinality of Reviews)*5

This is apparently too big comparing block nested loop join. Discard.

For merge sort join, we need to sort both business and review, the cost will be huge considering reviews is too big.

For Hash join, the cost is 3*3+3*22,950. This is apparently bigger than block nested loop join.

Thus, until here we use 22,967 IO + 689 IO = 23,656 IO

We will have 200(business)*90(average number of reviews per business) = 18,000 tuples.

Size per tuple: 52(size of projected business)+51(size of projected review)-10(duplicate bid after join)=93

Data pages: 93*18,000/4,000=419 pages

Have to write this out in the disk.

Now join Users with above which we call T for now:

For block nested loop join: use Users as outer

Cost: 419/100*14 = 59 IO

There is no index for both tables so index nested loops join is not an option.

For sort merge join, we will sort both tables;

To merge T, number of passes: $1+(\log_{99}(419/100))=2$ passes

This will result in more IO compared to block nested loop join which is only 59 IO.

For hash join, it would not be optimal as well. Since 3*Pages(Reviews)+3*Pages(T) is too large.

Thus, until here we use 23,656 IO + 419 IO = 24,075 IO

We will get 18,000*(1,825/90,000)=365 tuples And we project the required attributes as required:

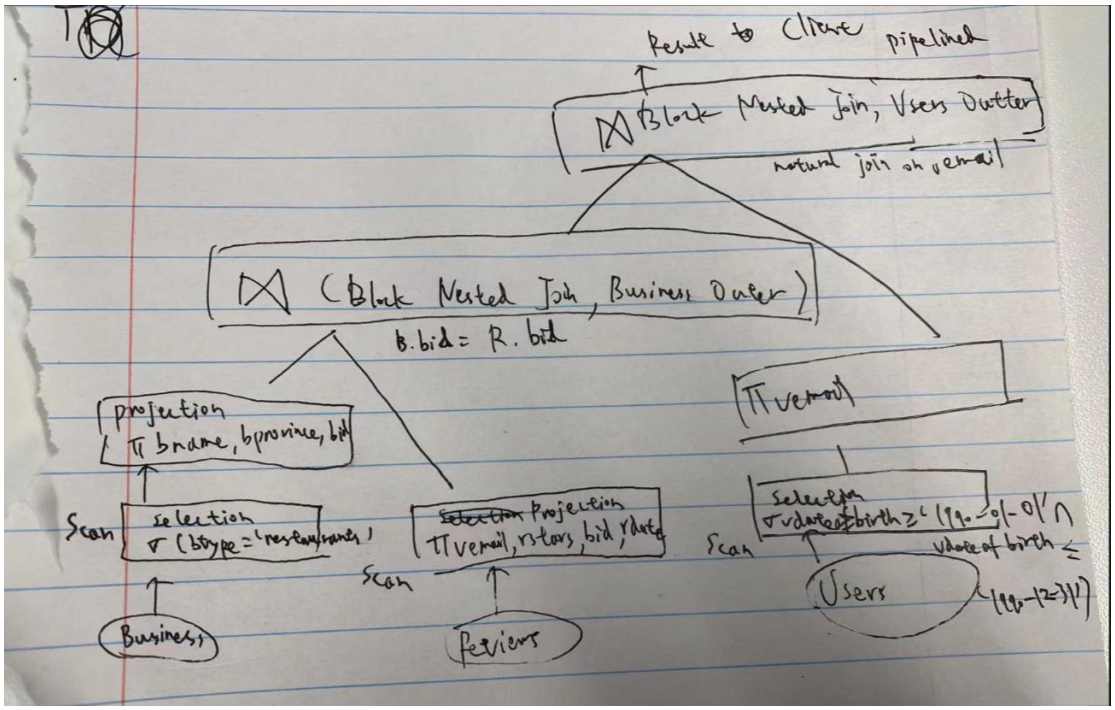
Bytes for each tuple:60*2/3+2+45*2/3+1+10=83 bytes

Total bytes: 83*365 =30,296 bytes

Pages required: 30,296/4000=8 pages We can give this to client. No IO cost.

Thus, until here total IO is 24,075 IO

I forgot that we need to scan the tree tables first to project each individuale tables. Thus adding that IO would be 24,075+275,400+2,460+947=302,882 IO



Ex.4

From reviews, we are selecting one month out of four years. We can first scan the table and project reviews in Dec.2021 with required attributes(rstars,uemail,bid).

31/(365*4)=reduction factors

31/(365*4)*1,800,000 = 38,220 tuples

38,220 tuples*(45*2/3+10+1)bytes / 4000 = 382 pages Have to write this to the disk

Call this intermidate relation R1.

Next, we do a join between R1 and users.

Using block nested join: 2,460+2,460/100*382 = 11,858 IO

There is no index for the two relations so index nested join not applicable.

Doing sorts for both relations will not be cheap, so we don't consider that.

Using hash join, we will read 2,460*3+382*3=8,526 IO

Until here the total IO is 8,526+382(projection)+275,400(scan reviews)=287,639 IO

Resulting tuples would be 38,220 tuples, bytes would be 10+1+82(rstars+bid+byets for user)=93 bytes, pages: 38,220*93/4000 = 889 pages

Now we project the required attributes, bytes would be (bid+uprovince+rstars+uemail)10+2+1+45*2/3=43 bytes, pages required: 38,220*43/4000 = 411 pages Write this to the disk.

Next we do a sort of uprovince,rid,uemail,rstars so that we can count and do average after.

We need Pass 0(5 runs), Pass 1(merge), cost of sorting is 2*2*411 = 1644 IO

We can remove the cost of last pass of sort by pipelining the output of Pass 1 to the aggregation. Cost would be 1,233 IO.

Thus we can pass this to produce groups based on bid and uprovince. In the aggregation, we keep a counter on rstars and distinct uemail and sum rstars as we do that. In the end we just need to calculate the average based on the sums of rstars/number of rstars.

The output is sent directly to client, so no IO cost.

So the total cost is 287,639+1,233 = 288,872 IO