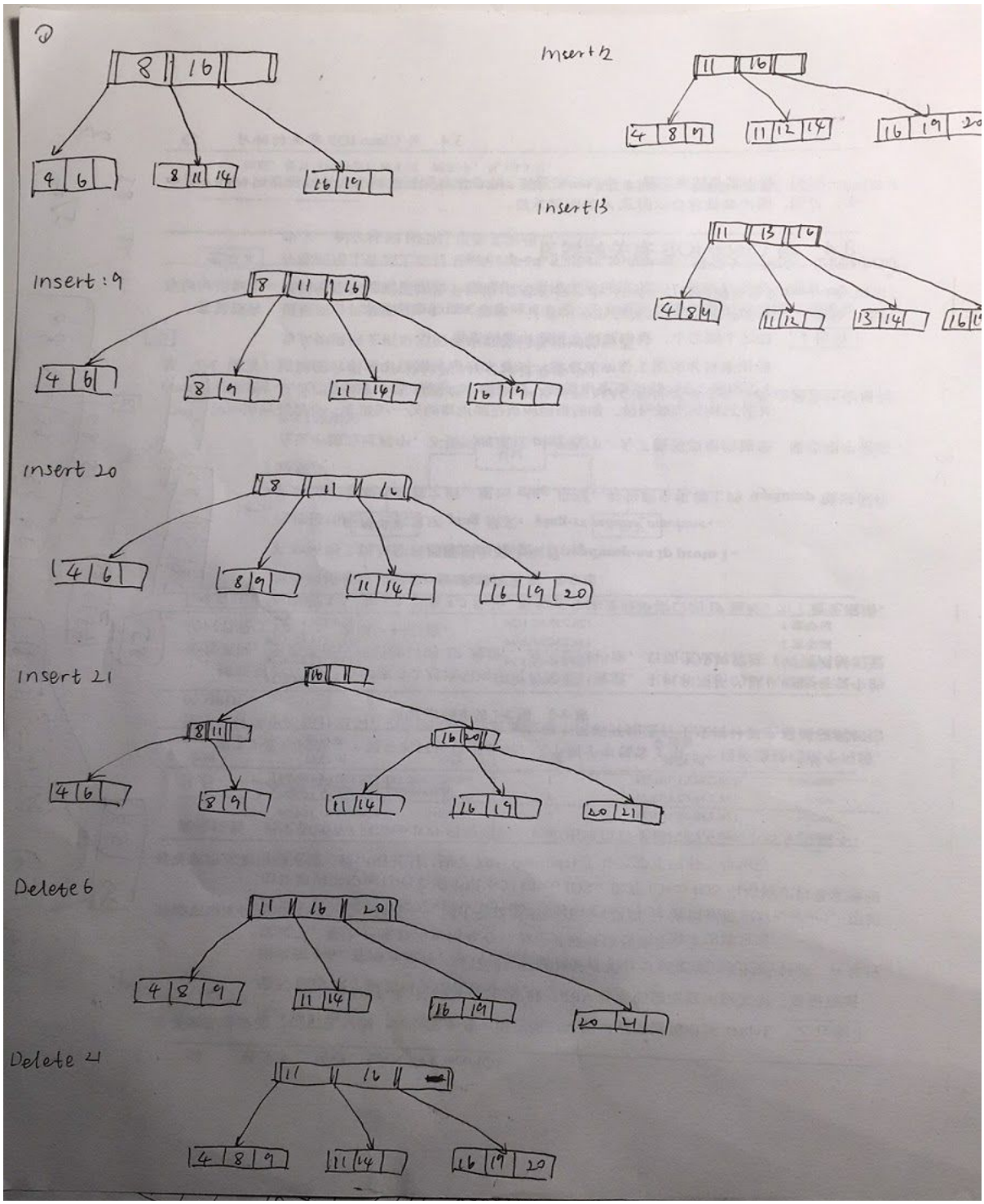
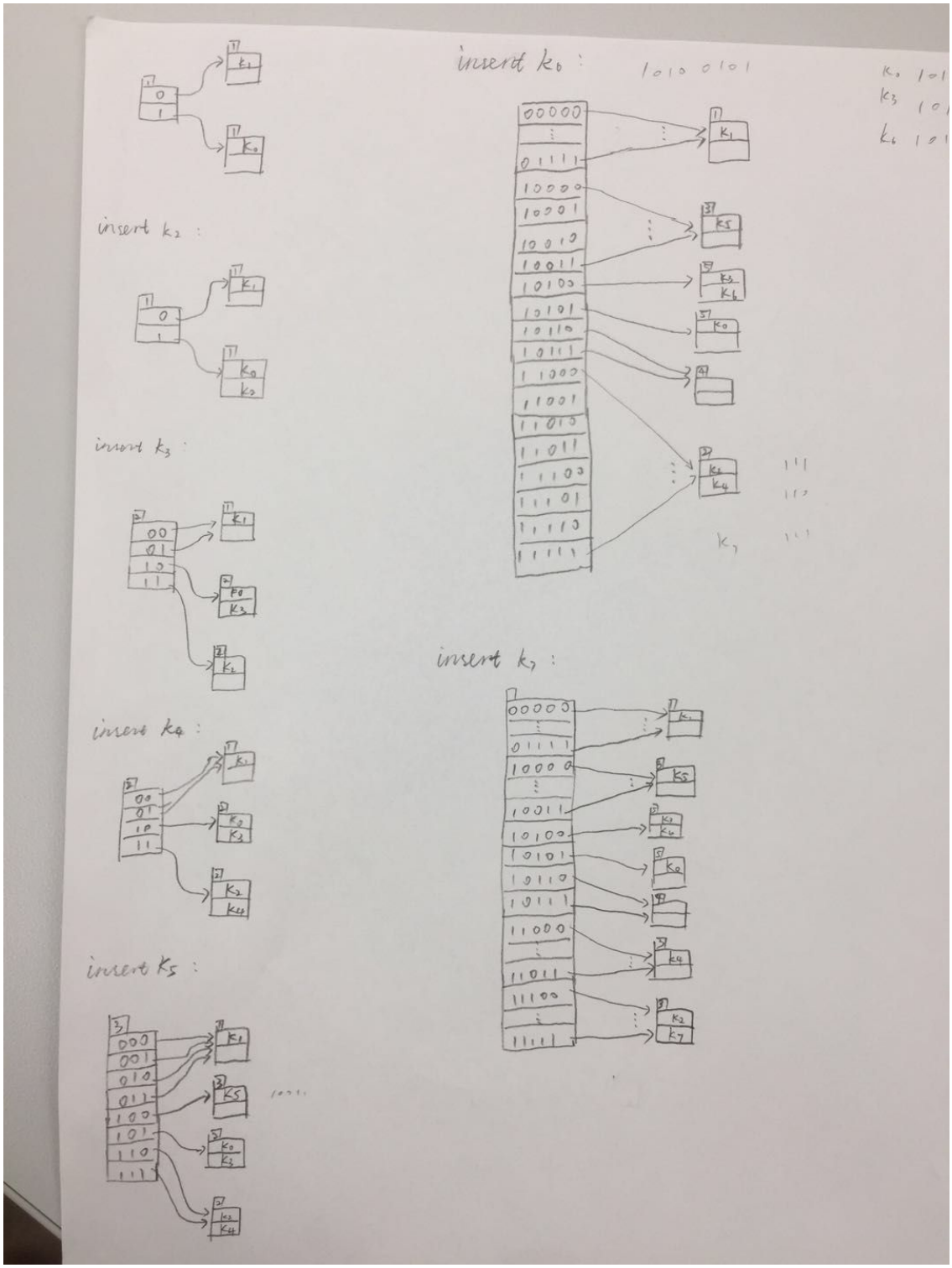
Database Homework 4

Shuai Zhang, sz1950

1.



2.



3.

1.

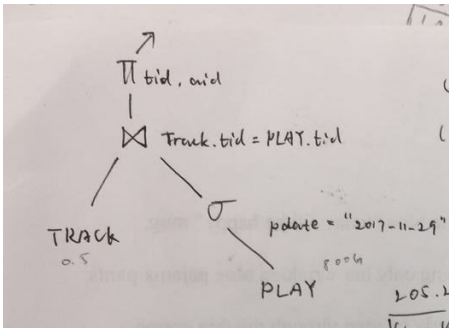
q1.List all tracks' id with it's artist's id that played on 2017-11-29.

q2.List all tracks id with it's artist's id that compose by artists whose genre is jazz and played on 2017-11-29.

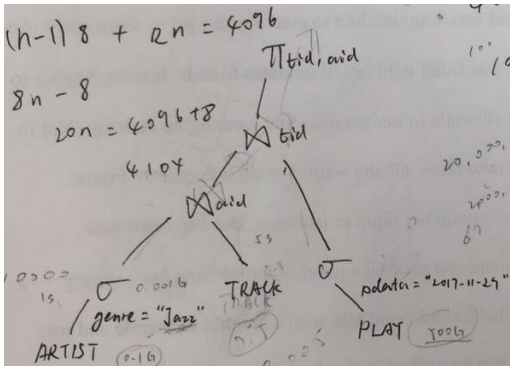
q3.List all user who live in Miami and have ever play tracks that has duration less than 10 second.

2.

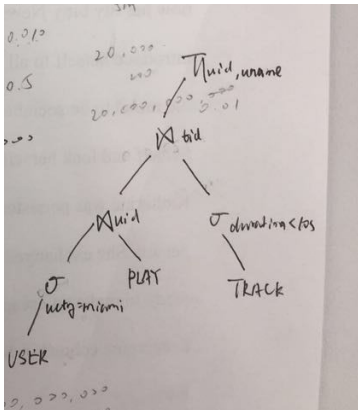
q1.



q2.



q3.



(b)
q1.
TRACK has 0.5G of data which fit in the memory. Then we scan PLAY and filter with pdate =”2017-11-29”, and join them by tid. At last, we output the selected tid and aid. PLAY has 800G data. So, the total amount we need to read from disk is 0.5G + 800G = 800.5G. Thus the usage of time will be 8005s.

q2.
First, we scan ARTIST with genre equals Jazz which has 0.1G data, 1,000,000 tuples. Then we scan TRACK which has 0.5G. On average, every artist has 5 tracks. 1% artist play jazz, which is 10,000 tuples, so After the join, 50,000 tuples will be selected and it takes 0.005G places that can be hold in memory. At last, we scan the PLAY table which has 800G. So, the total disk operation will be 800G + 0.5G + 0.1G = 800.6G, The time consumption will be 8006s.

q3.
First, we scan USER, which is 10G, and select users live in Miami (0.1%). The result will be 0.01G, 100,000 tuples of users. Each user plays 200 tracks on average. The table after join on USER and PLAY will has 20,000,000 tuples, which is 20M*40B = 0.8G that fit in memory. At last we scan the TRACK, which is 0.5G, and get track with duration less than 10s. So, the total disk operation is 10G from USER, 0.5G from TRACK, 800G from PLAY, 800G + 10G + 0.5G = 810.5. Thus the time will be 8105s.

(c)
Unclustered index on uid in the Artist table:

Each tree node contains n-1 keys and n pointers. With 8(16) bytes per key, 12(8) bytes per pointer and a node size of 4096 bytes, we can find how many keys and pointers can fit in each node: (n-1) * 8 + 12 * n = 4096 => n = 205. Assuming 80% occupancy per node, each leaf node will contain about 164 index entries and each internal node will have 164 children.

For the Artist table, there's a total of 1,000,000 index entries. Since about 164 index entries are in each leaf node there will be about 1,000,000/164 ~ 6097 leaf nodes. On the next level there will be about 6097/164 = 37 nodes, and then we have the root of the tree. So the B+ tree has 3 levels of nodes. Thus is takes about 4 * 5 ms = 20 ms to fetch a single record from the table using this index assuming no caching. The size of the tree is dominated by the leaf level, which is about 6097 * 4KB ~ 24.388MB.

Dense unclustered index on tid in the Play table:
Each index entry now has a 8-byte uid, and an 12-byte RID, and thus takes 44 bytes. Thus n ~ 205, and with 80% occupancy we get about 164 entries per node.

There are 20 billion index entries, and thus there are 122 million nodes at the leaf level, 743920 million nodes at the next level, then 4536, then 27, then the root. Thus the tree has 5 levels of nodes and 6 * 5 ms = 30ms is needed to fetch a single record from the table. The cost of fetching 50 records would involve 49 additional seeks into the underlying table, adding 49*5 + 30= 275ms for a single record. The size of the tree is dominated by the leaf level, which is about 122 billion * 4KB = 488GB.

(d)
q1.
We would choose a clustered index on pdate. Now it will take 80s on query play table. The total is now 85s.

q2.
We would choose a clustered index on pdate in PLAY and gerne in ARTIST. Now the total will be 80s + 5s + 0.01s = 85.01s.

q3.
We would choose a clustered index on uid in PLAY and ucity in USER. Then we need 0.1% time of origin query on user which is 0.1s. Then we join PLAY with USER with select uid, so we only need 0.1% time of the origin scan of PLAY which is 8s. The total will be 8s + 0.1s + 5s = 13.1s.

4.
(a)
The capacity of the disk is: 3*200,000*2000*1024 bytes = 614,400 MB.
The maximum rate of a read is 200*1000*1024 byte/s ~ 100 MB/s
The average rotational latency is 2.5ms.

(b)
Block model:
Read 4KB: t = 4+2.5+4/200ms ≈ 6.52ms
Read 20KB: T = 5t = 50.2ms
Read 200KB: T = 50t = 326ms
Read 20MB: T = 5000t = 32.6s
Read 2G: T = 500000t = 3260s

LTR model:
Read 20KB: T = 6.5 + 200/200ms = 7.5 ms
Read 200MB: T = 6.5 + 20000/200 = 106.5 ms
Read 20MB: T = 6.5 + 2000000/200 = 10006.6 ms

(c)
phase 1:
Divide into 256/4 = 64 blocks to sort.
Time for IO ~ 2560s

phase 2:
merge the 64 blocks in phase 1.
The main memory is divided into 65 buffers, 64 read plus 1 write.
Each buffer is of size 4GB/65 = 61.54 MB.
I/O for each buffer: 6.5ms + 61.54/200s = 314.2ms.
256GB / 61.54MB = 4260 pieces.
IO for buffer: 0.3142s * 4260 * 2 = 2677s
Total time: 2677s + 2560s= 5236s.

(d)
We can use 8 way merge. Then we have 8 + 1 = 9 buffers. Each buffer has size 444MB. IO for each buffer will be 6.5 + 444/200 = 2.2s. Reading all 256GB will take very close to 2560 seconds, since the cost of seeks is very small compared to the cost of transfer. The same is also true for 10-way, 5-way, and 16-way merges, so the two options have almost exactly the same cost. A precise analysis would show the first

option to be slightly better, but all options are worse than the one taken in part (c) where we have a single merge. (Note also that the cost of, say, a 5-way merge step does not depend on the ordering of the merge steps, so a 5-way followed by a 16-way has the same cost as 16-way followed by 5-way.