Question 1. Answer the following questions.

[25 marks]

A relation called *employee* holds 30,000 tuples. Each tuple is stored as fixed length and fixed format record. Each tuple has the length of 250 bytes and the key attribute *ID* has the length of 20 bytes. The tuples are stored sequentially in a number of blocks, ordered by the *ID* attribute. Each block has the size of 4,096 bytes (i.e., 4 Kilobytes). Consider creating a primary index on the *ID* attribute, and a secondary index on the *city* attribute (stores the city where an employee is from). Each index entry consists of a search key and a 10-byte long pointer to data. Assume that a record or an index entry can only be stored in one block.

a) Compute the number of disk blocks needed to store the relation *employee*.

[4/25]

b) If the primary index on *ID* is sparse (i.e., one index entry for one block), compute the number of blocks needed to store the index.

[4/25]

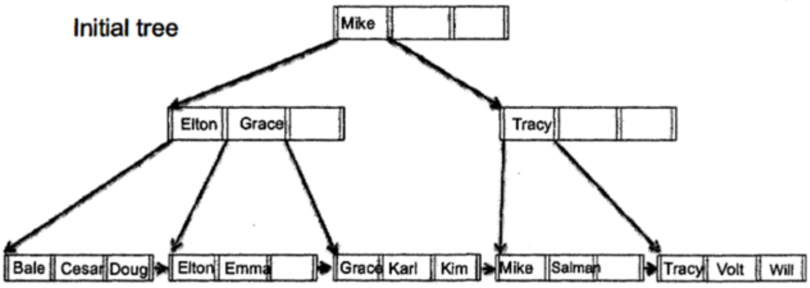
c) Is a sparse secondary index on the *city* attribute useful? Why?

[3/25]

d) Suppose that the number of pointers in a B+ tree node is 20 and there are one million distinct search key values to be indexed. What is the maximum height of the B+ tree?

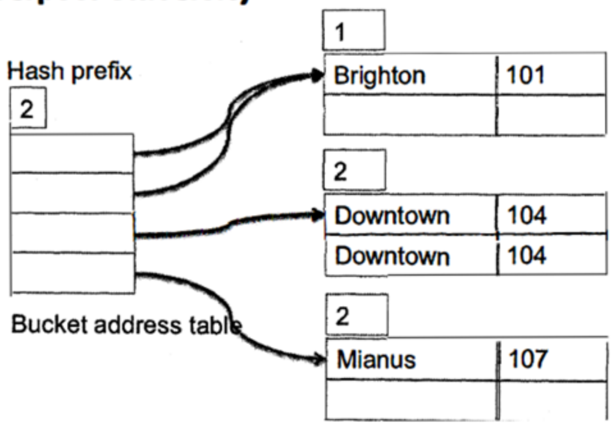
[2/25]

e) The initial B+ tree for indexing *names* in the *employee* is shown below. Draw the B+ trees after the following operations: (1) insert Amy; (2) insert Linda; (3) delete Salman. Each subsequent operation should be performed based on the result of the previous operation.



[9/25]

f) An extendable index (see diagram below) is created on the *city* attribute. Describe in detail how the index will be updated if another two records with the search key value of "Mianus" are inserted.


Hash prefix
2
Bucket address table
1
Brighton
101
2
Downtown
104
Downtown
104
2
Mianus
107

[3/25]

Question 2. Consider the following two relations and their catalogue information:

i) *account(account_Number, customer_Name, balance, branch_ID)*

ii) *branch(branch_ID, branch_Name, branch_City, postcode)*

The "*account_Number*" is the key for the *account* relation, and the "*branch_Name*" is the key for the *branch* relation. The *account* relation contains 300,000 records stored in 60,000 blocks, and the *branch* relation contains 500 records stored in 50 blocks. Assume that both relations are sequentially stored by the key attributes. Answer the following questions.

[25 marks]

a) Suppose that the linear search algorithm is used to evaluate the selection $\delta_{balance>5,000}$ in the *account* relation, how many block transfers are needed? How many seeks are needed?

[4/25]

b) Suppose that a sparse B+ tree index (with the number of pointers in a node, $N=7$) has been created for the *account* relation on the attribute "*account_Number*", how many block transfers are needed for the selection $\delta_{account\_Number\ =11737}$? How many seeks are needed? (Hint: in a sparse index, every index entry contains a pointer to one block)

[4/25]

c) Suppose that none of the relations can fit in memory, and the nested loop join algorithm is used to evaluate "*account* $\bowtie$ *branch*". Which relation should be used as outer relation? How many block transfers are needed? How many seeks are needed?

[5/25]

d) Suppose that the external sort merge algorithm is used to sort the *account* relation on the *account_Number* attribute. Assume that the memory size $M=30$ and the buffer for reading and writing $b_b=2$. How many block transfers are needed? How many seeks are needed?

[4/25]

e) Suppose that the hash join algorithm is used to evaluate "*account* $\bowtie$ *branch*", the number of partitions, $n_h=70$, and the size of the buffer for reading and writing, $b_b=2$. How many block transfers are needed? How many seeks are needed?

[4/25]

f) With regard to the results obtained from Questions 2.c) and 2.e), discuss which algorithm is more efficient in evaluating the join.

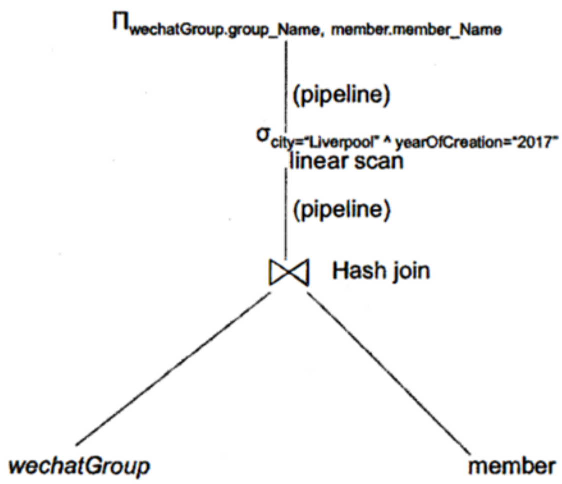
[4/25]

Question 3. Consider the following two relations and their catalog information. Answer the questions below.

wechatGroup(<u>gourp_ID</u>, group_Name, size, yearOfCreation, owner)
<u>member</u>(<u>member_ID</u>, member_Name, city, group_ID)

- *group_ID* and *member_ID* are the keys for the two relations, respectively.
- number of tuples in relation *wechatGroup*, n_r = 1,000
- number of blocks in *wechatGroup*, b_r =200
- number of distinct values on attribute *yearOfCreation* in *wechatGroup*, V(*yearOfCreation*, *wechatGroup*) = 100
- number of tuples in relation *member*, n_s = 10,000
- number of blocks in relation *member*, b_s =1,000
- number of distinct values on attribute *city* in *member*, V(*city*, *member*) = 100

A query evaluation plan is shown below.



[25 marks]

a) What is the relational algebra expression for the given evaluation plan?

[3/25]

b) One of the heuristic rules for query optimisation is to perform selection operations as early as possible. Write the equivalent algebra expression for the answer from Question 3.a).

[4/25]

c) What are the differences between materialisation and pipelining in evaluating an expression?

[4/25]

d) Suppose that all selections are evaluated by using linear scan and pipelining is used for projection. Draw an annotated evaluation tree for the relational algebra expression obtained from Question 3.b).

[4/25]

e) Based on the catalog information, what is the estimated size of the selection $\sigma_{yearOfCreation="2017"}(wechatGroup)$?

[4/25]

f) Assume that for each tuple in $\sigma_{yearOfCreation="2017"}(wechatGroup)$, the average cost of performing the hash join is 3 block transfers. What is the total number of block transfers for the whole evaluation plan in Question 3.d)?

[6/25]

Question 4. Answer the following questions.

[25 marks]

a) Draw a precedence diagram for the following schedule. Is it conflict serialisable?

T1	T2	T3	T4
read(Y)			
read(X)			
write(X)			
		write(Y)	
	read(X)		
	write(Y)		
			read(W)
			write(W)
			read(Y)
	read(W)		

[4/25]

b) Is the following schedule recoverable? Justify your answer.
Schedule: T1:write(X); T1:write(Y); T2:read(X); T2:write(Y); T2:read(Z); T1:write(Z); T1:commit; T3:read(Y); T2: commit; T3:write(Z); T4:read(Z); T3:commit; T4:abort.

[4/25]

c) Consider the following schedule. Assume that the database failure happens at time=18, answer the following questions: (1) at the checkpoint, what transactions are in the list ***L***? (2) Which transactions need to be redone? (3) Which need to be undone? Justify your answer.

Time	T7	T8	T9
0			start
1			read(B)
2	---------------------------**Checkpoint {*L*}**---------------------		
3			B=B+10
4	start		
5	read(A)		
6	A=A+1		
7			write(B)
8			commit
9		start	
10		read(A)	
11		read(B)	
12		B=A+2B	

13		write(B)	
14		commit	
15	read(B)		
16	B=B+3A		
17	write(A)		
18	-------------------------System Failure-------------------------		

[5/25]

d) Describe how the centralised deadlock detection method is used to detect deadlocks in a distributed database system.

[6/25]

e) In the context of distributed database, briefly describe how the "primary site" and "peer-to-peer" methods work for asynchronous data replication, respectively.

[6/25]

END OF EXAM PAPER