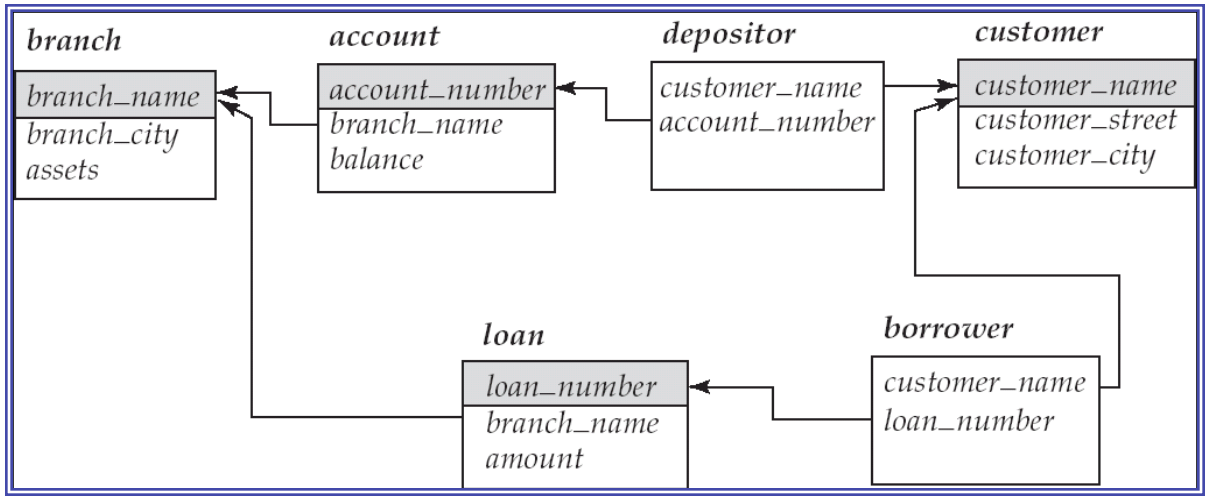
Database Development and Design (CPT201)

Tutorial 4

Dr. Wei Wang

Department of Computing

Q1



- Given the following information:
  - Number of records of *customer*: 1,000,000
  - Number of blocks of *customer*: 8,000
  - Number of records of *depositor*: 500,000
  - Number of blocks of *depositor: 1,000*
- Assume we use indexed nested loop join for evaluating the join; and use the depositor as outer relation. We also assume that customer have a primary B+-tree index on the join attribute customer-name, with n=100.
  - What is the number of block transfers and seeks, respectively?
  - Assume t_T=0.1ms and t_S=4ms, what is the total cost in terms of time?
  - Together with the nested loop join and block nested loop join, which method is the best in terms of total cost?

Q1 Solution

- We use $b_r + n_r * c$, where $c = h_i + 1$
- In time: $b_r (t_T + t_S) + n_r * c$, where $c = (h_i + 1) * (t_T + t_S)$
- Answers:
  - Number of block transfer: 1000+(4+1)*500,000; number of seeks is the same
  - Cost in terms of time: 2,501,000*(4+0.1)ms. In total 10,254,100ms
  - Which one is the best
    - Nested loop: 500,000*8,000+1,000 (block transfers); 500,000+1,000 (seeks). In total 402,004,100ms
    - Block nested loop: 1,000*8,000+1,000(block transfers); 2*1,000(seeks). In total 808,100ms
    - So "block nested loop" is best in this example.

Q2

- Consider the following two relations and their catalogue information.
  - i) *customer(customer_Number, name, phone, address)*
  - ii) *transaction(transaction_ID, date, amount, customer_Number)*
- The "*customer_Number*" is the key for the *customer* relation, and "*transaction_ID*" is the key for the *transaction* relation. The *transaction.customer_Number* is the foreign key referencing the *customer* relation. The *customer* relation has 5,000 tuples stored in 500 blocks, and the *transaction* relation contains 80,000 tuples stored in 7,000 blocks. Assume that the relation *transaction* has been sequentially sorted by the key attribute. Answer the following questions.

Q2 cont'd

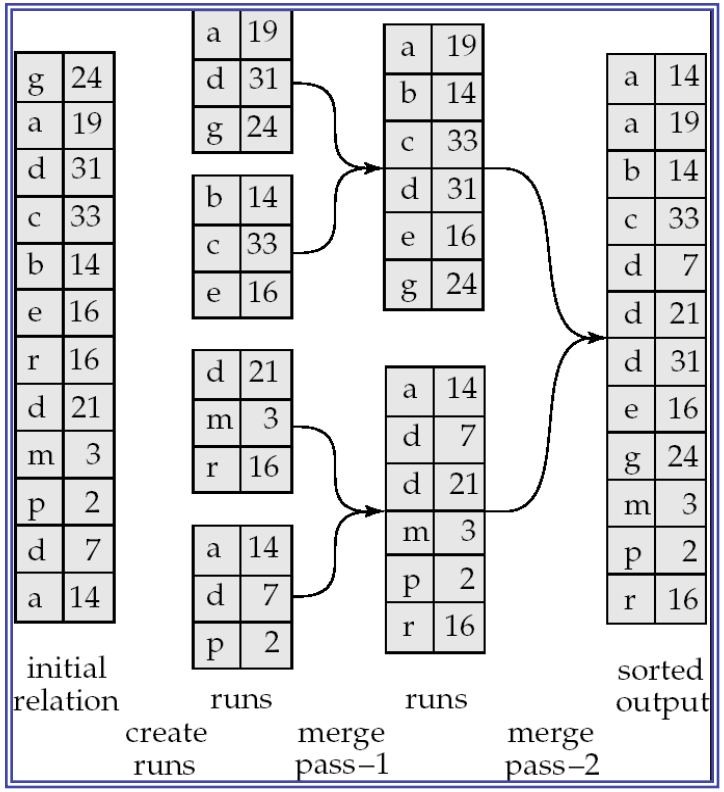
- Assume that none of the relations can fit in memory, time for one block transfer t_T=0.1ms, and time for one seek t_S=4ms.
- Suppose that a sparse primary B+ tree index (*N*=7) has been created for the *transaction* relation on the key attribute.
- Which of the two algorithms, the block nested loop join and indexed nested loop join, is more efficient to evaluate "*customer transaction*"? Justify your answer.

Q2 Solution

- Using block nest loop join: customer (smaller relation) should be used as the outer relation.
  - Number of block transfers: $b_r*b_s+b_r$ = 500*7,000+500 = 3,500,500
  - Number of seeks: $2b_r$ = 2*500 = 1,000
  - In total, it would take 3,500,500*0.1+1,000*4 = 354,050ms.
- Using indexed nest loop join: customer (smaller relation) should be used as the outer relation.
  - Height of the tree at most is 7.
  - Number of block transfers: n_r*c+b_r = 5,000*8+500 = 40,500
  - Number of seeks: 40,500
  - In total, it would take 40,500*0.1+40,500*4 = 166,050ms.
- So the indexed nested loop join is more efficient.

Q3

This is about the external merge sort. Suppose that Memory hold at most three page frames, only one tuple fits in block. The relation needs 12 blocks to store, m= 3, *assume* b_b =1.



Total block transfer = ?

Total seeks = ?

Q3 Soution

- External merge sort: $b_r (2 \lceil \log_{M-1}(b_r / M) \rceil + 1)$ block transfers and $2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{M-1}(b_r / M) \rceil - 1)$ seeks
- So block transfers: $12(2\log_2 4 + 1) = 60$
- Seeks: $2*12/3 + 12/1(2\log_2 4 - 1) = 44$

Q4

- Given the following information. Assume relations are not sorted (need to be sorted using external merge sort).
  - Number of records of customer: 1,000,000
  - Number of blocks of customer: 8,000
  - Number of records of depositor: 500,000
  - Number of blocks of depositor: 1,000
- If we use merge join, assume memory size M=100 and the number of blocks allocated in memory for buffering each relation is $b_b=5$, what is the cost in terms of block transfer and seek?

Q4 Solution

- Merge join $b_r + b_s$ block transfers +
  $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ seeks
- So it needs 9,000 block transfers and 1,800 seeks
- However, we need to consider the cost for sorting both relations as they are not sorted. External merge sort: $b_r (2 \lceil \log_{(\lfloor M/bb \rfloor -1)}(b_r / M) \rceil + 1)$ block transfers and $2 \lceil b_r / M \rceil + \lceil b_r / b_b \rceil$ $(2 \lceil \log_{(\lfloor M/bb \rfloor -1)} (b_r / M) \rceil -1)$ seeks
- So merge sort for both relations needs block transfers of $8000(2\log_{(100/5-1)}(8000/100)+1)$ and $1000(2\log_{(100/5-1)}(1000/100)+1)$, that is 40,000 and 3,000, in total 43,000 block transfers.
- It needs seeks $2*8000/100+8000/5*(2\log_{(100/5-1)}(8000/100)-1)$ and $2*1000/100+1000/5*(2\log_{(100/5-1)}(1000/100)-1)$, that is 4,960 and 220, in total 5,180 seeks.
- However, as we do external merge-sort and then merge-join, the cost for last writing should be added. That means another 9,000 block transfers and 1,800 (or 9,000/5) seeks.
- **So, in total we need 61,000 block transfers and 8,780 seeks.**

Q5

- Given the information from Q3,
- *depositor* is to be used as build input. Partition it into 50 partitions, each of size 20 blocks. This partitioning can be done in one pass. Similarly, partition *customer* into 50 partitions, each of size 160. This is also done in one pass.
- Assuming 20 blocks are allocated for the input buffer and each output buffer (b_b=20)
- What is the total cost (the cost of writing partially filled blocks needs to be considered)?

Q5 Solutions

- We use the fomulae: $3(b_r + b_s) + 4 * n_h$ block transfers + $2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil) + 2 * n_h$ seeks.
- So 3*(1000+8000)+4*50 block transfers and 2(1000/20+8000/20) +2*50.
- That is 27,200 block transfers and 1,000 seeks.

Q6

- Consider the following two unary relations *r*, b_r=5 and *s*, b_s=3.
- Assume that each block can hold two tuples from r and s; the memory size is 3, and one block is used to buffer the read/write, i.e., b_b=1. Note that if a relation can fit into memory, the external merge sort is not needed.
- Answer the following questions on external merge sort and merge join.
  - (a) How many sorted runs will be produced for relations r and s, respectively?
  - (b) How many merge passes are needed to sort r and s, respectively?
  - (c) Assume that the cost for writing the final results to disk needs to be considered in order to perform the subsequent merge join, what are the numbers of block transfers for sorting r and s, respectively?
  - (d) What are the numbers of block transfers for the merge join of r and s?
  - (e) What is the total cost in terms of block transfers for the whole process?

Q6 Solution

- (a) br=5; bs=3, so number of sorted runs for r is 5/3, take the upper value which is 2. The first run file has 3 blocks and the second run file has 2 blocks. For s, NO run file needs to be generated as s can fit into memory.
- (b) r: $\log_2(5/3)\approx1$, so the number of merge passes is 1. For s, no need to use external merge sorting.
- (c) using the formulae for calculating the number of block transfer. But we need to add the cost of writing the result to disk, which is the number of b_r and b_s. For r: $br(2(\log_{(M-1)}(br/M)+1)+br = 5*(2*\log_2 5/3+1)+5 = 20$. For s: just perform the in-memory sorting, 2*3=6
- (d) for merge join of r and s, 5+3=8 block transfers are needed.
- (e) number of block transfers: 20+6+8=34 block transfers.