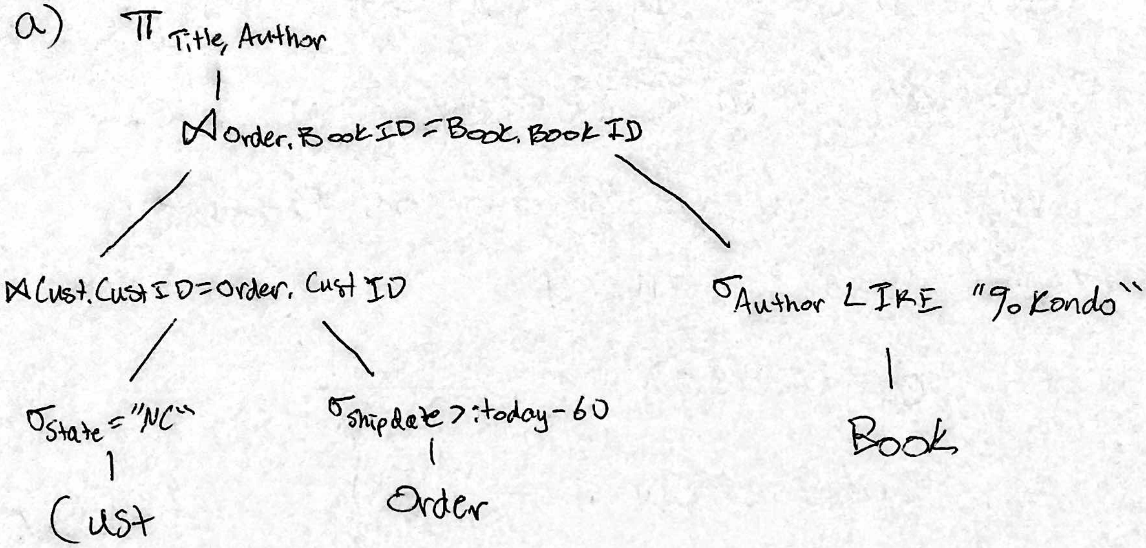
3

a)



b) $\text{Tuples/Shipday} = \frac{|\text{Order}|}{|\pi_{\text{shipdate}}\text{Order}|} = \frac{60{,}000}{1{,}000} = 60$

60 days in range >: today -60

$60 \cdot 60 = 3{,}600$ tuples

c) Assuming preservation of value sets,

$|\pi_{\text{cust ID}}(\sigma_{\text{shipdate} >: \text{today}-60}\ \text{Order})| = |\pi_{\text{custID}}\text{Order}| = 3{,}000$

d) $|\sigma_{\text{state}=\text{"NC"}}(\text{customer})| = \frac{|\text{Cust}|}{|\pi_{\text{state}}(\text{cust})} = \frac{3000}{50} = 60$

e) $= \frac{|\sigma_{\text{state}=\text{"NC"}}\text{Customer}| \cdot |\sigma_{\text{shipdate} >: \text{today}-60}\ \text{order}|}{|\pi_{\text{custID}}\text{Order}|}$

$= \frac{60 \cdot 3600}{3000}$

$= 72$

f) The best execution plan is to Scan Order from the ~~disk~~ and then filter it by (shipdate = :today). You also want to scan Inventory from the disk as well and use a hash join on BookID, since that uses the least memory.

The I/O cost of Scan(Order) is $\frac{|order|}{10 \text{ rows/block}} = 6{,}000$ blocks,

The I/O cost of Scan(Inventory) is $\frac{|Inventory|}{10 \text{ rows/block}} = 4{,}000$ blocks

Filtering ~~by~~ order by shipdate results in 60 entrees which is 6 blocks and can be stored in memory.

In the hash join we can use these 6 filtered blocks of order and stream in the Inventory blocks we scanned to probe and join, which doesn't require any additional I/O's, so the total I/O cost is 6,000 + 4,000 = 10,000 I/O blocks.

g) The best plan now is to use an Index nested-loop join. We scan in Order and filter it still, but then we use the filtered blocks order.BookID to probe the index on Inventory.BookID to output pairs matching on BookID.

The I/O cost of scanning Order is still 6,000 I/O blocks, producing 6 filtered blocks with 60 rows.

The B+ tree index lookup cost for Inventory is at most 6 blocks (assuming no optimization since it isn't necessary) since there are 40,000 rows which becomes a root node level, an intermediate node level, and a leaf node level with up to 4 blocks to search through (for 40 rows).

Thus performing an index lookup for the 60 filtered rows from order produces at most 6·60 = 360 I/O's for a total I/O cost of 6,360 I/O blocks.