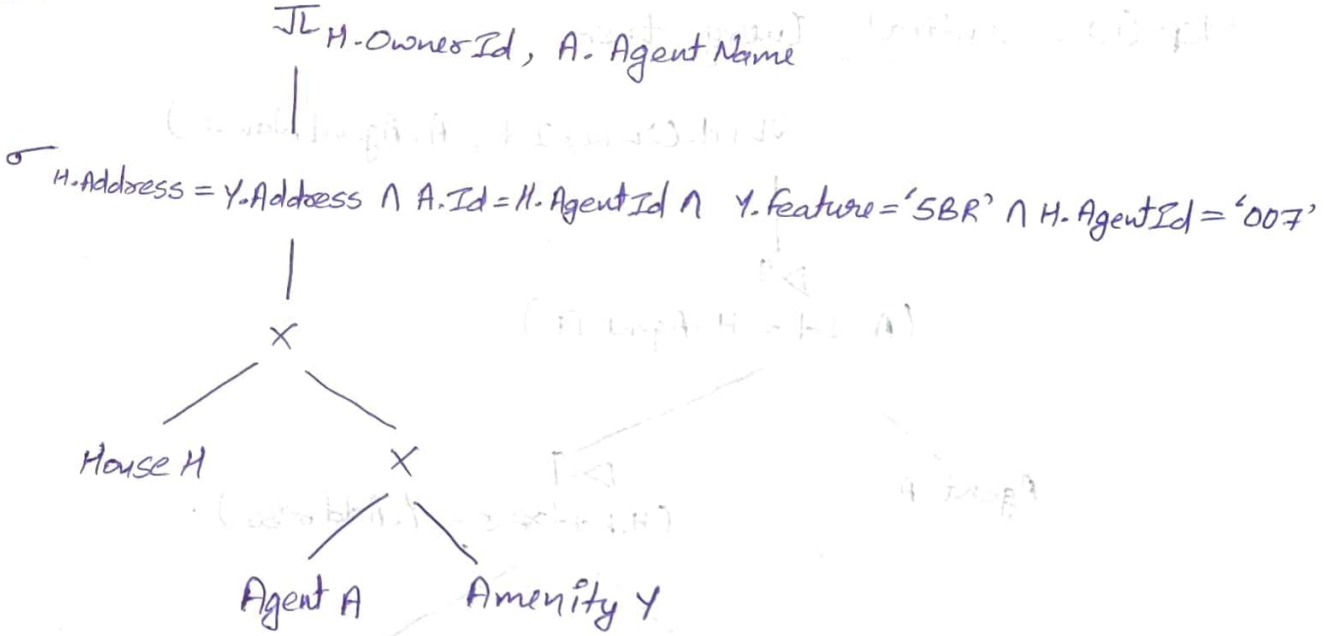
Answer 1:

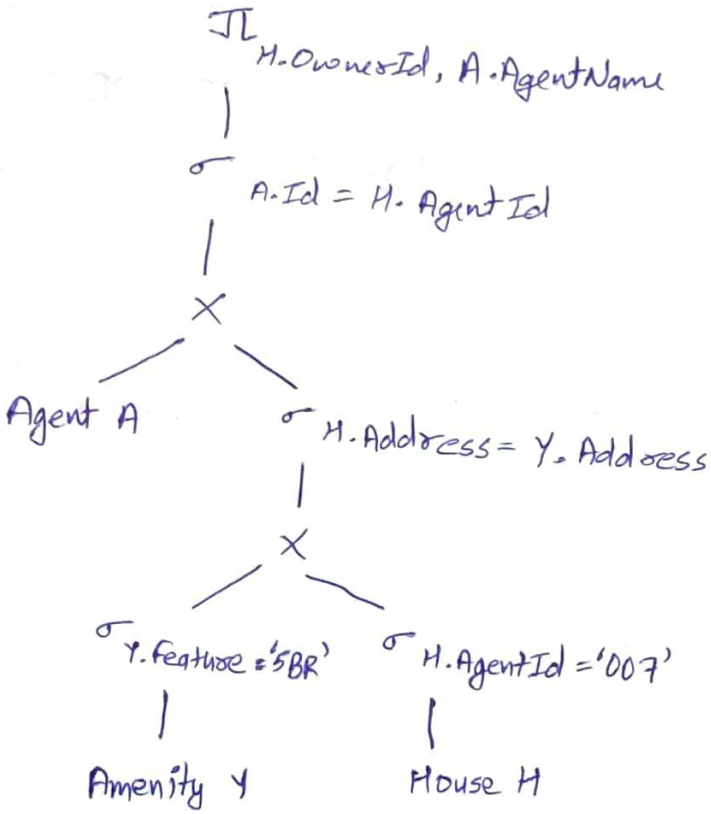
SELECT H.OwnerId, A.AgentName FROM House H, Agent A, Amenity Y WHERE H.Address = Y.Address AND A.Id = H.AgentId AND Y.Feature = '5BR' AND H.AgentId = '007';

$$\Pi_{H.OwnerId,\ A.AgentName}\left(\sigma_{H.Address = Y.Address \wedge A.Id = H.AgentId \wedge Y.Feature = '5BR' \wedge H.AgentId = '007'}(House\ H \times Agent\ A \times Amenity\ Y)\right)$$

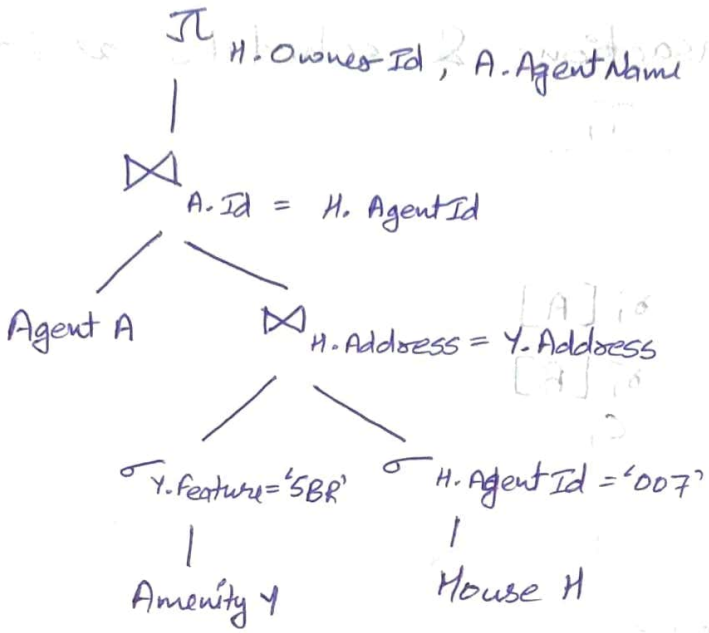
Step 1



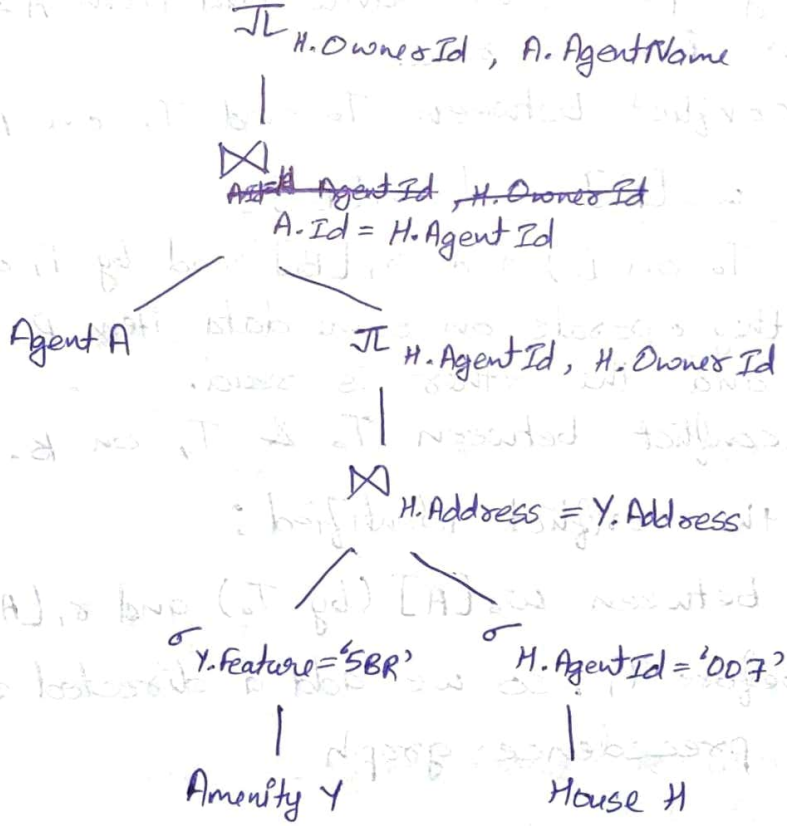
Step 2 : Moving Select operation down, and rearrange leaf



Step 3: Combine Cartesian Product and select into join



Step 4: Moving Projection Down



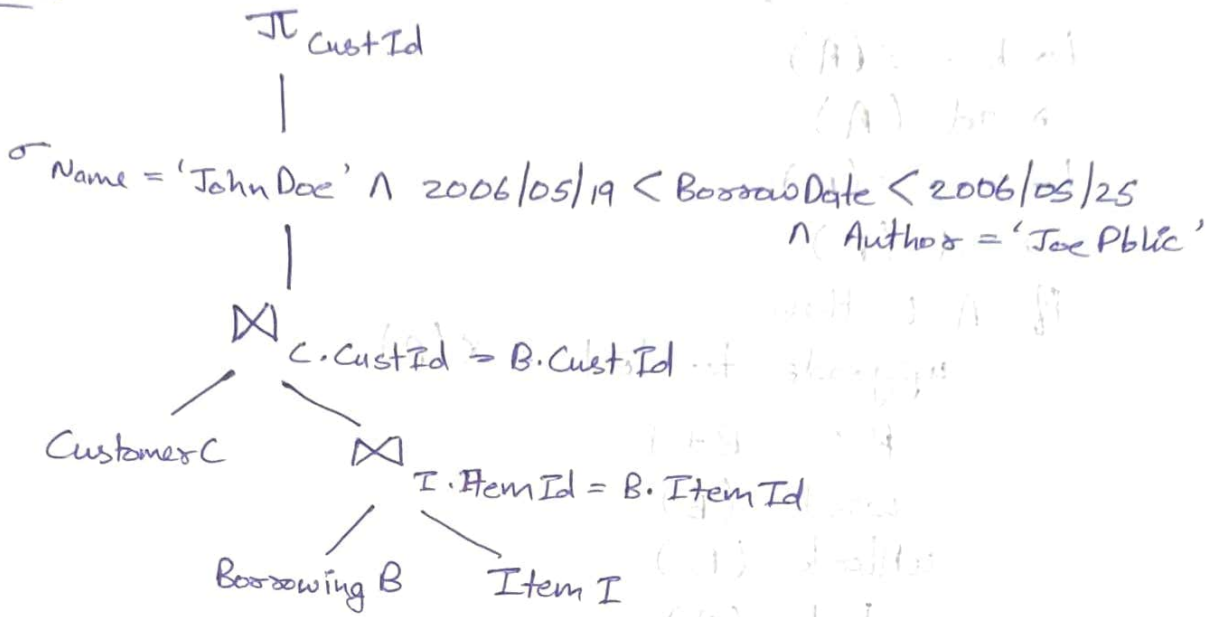
Final Expression –

$$\pi_{H.OwnerId,\ A.AgentName}\left(\text{Agent A} \bowtie_{A.Id = H.AgentId} \left(\left(\sigma_{Y.Feature='SBR'}(\text{Amenity Y})\right) \bowtie_{H.Address = Y.Address} \left(\sigma_{H.AgentId='007'}(\text{House H})\right)\right)\right)$$

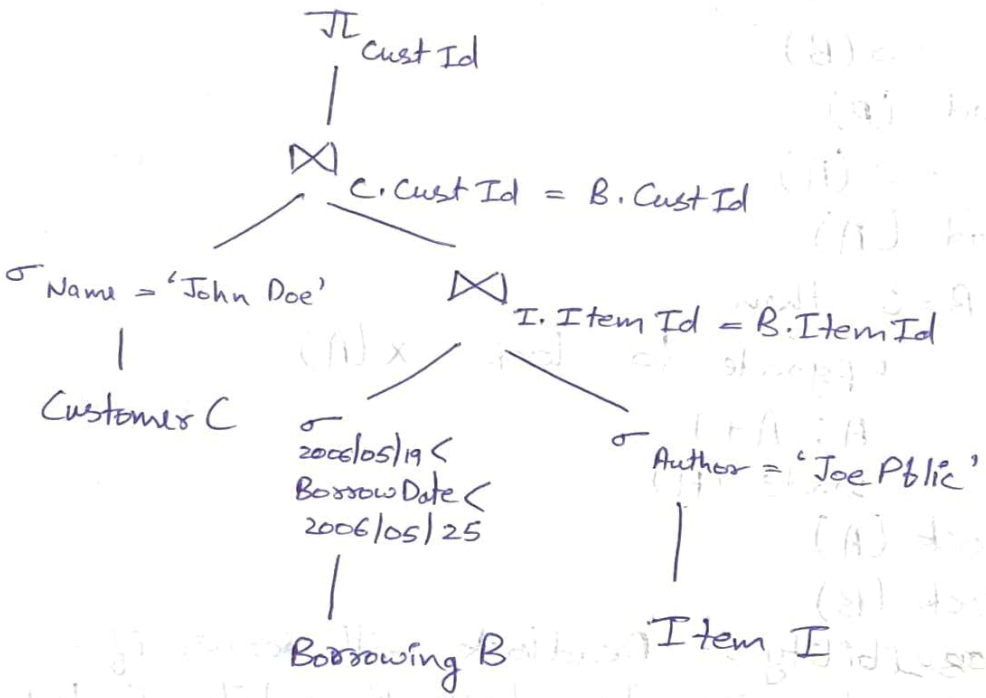
Answer 2:

$\Pi_{\text{Cust Id}}(\sigma_{\text{Name} = \text{'John Doe'} \text{ AND } 2006/05/19 < \text{BorrowDate} < 2006/05/25 \text{ AND } \text{Author} = \text{'Joe Public'}}(\text{Customer} \bowtie \text{Borrowings} \bowtie \text{Item}))$

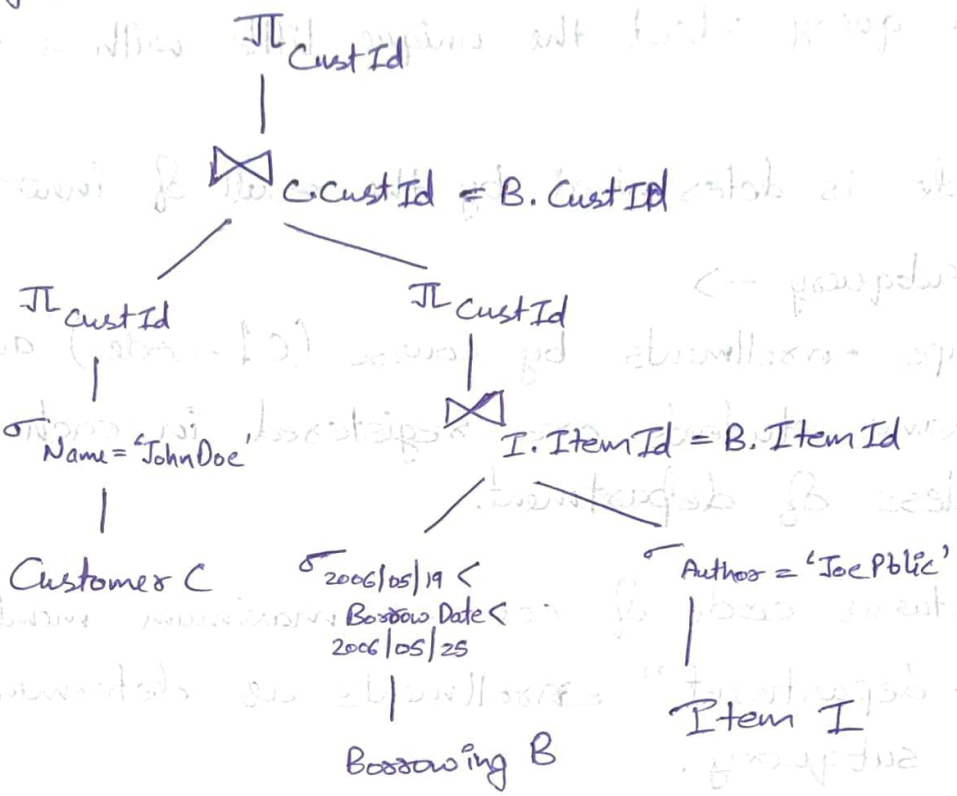
Step 1:



Step 2: Move Select down



Step 3:-
Move Projection down



$$\Pi_{CustId}\left(\left(\left(\Pi_{C.CustId}\left(\sigma_{Name='JohnDoe'}(Customer\ C)\right)\right)\bowtie_{C.CustId=B.CustId}\left(\Pi_{B.CustId}\left(\left(\sigma_{2006/05/19<BorrowDate<2006/05/25}(Borrowing\ B)\right)\bowtie_{I.ItemId=B.ItemId}\left(\sigma_{Author='JoePblic'}(Item\ I)\right)\right)\right)\right)\right)$$

SQL

```
SELECT C.CustId FROM(
   (SELECT C.CustId FROM Customer C WHERE C.Name = 'John Doe')
   JOIN
   (SELECT B.CustId
    FROM (
          (SELECT * FROM Borrowing B WHERE BorrowDate > '2006/05/19'
                                       AND BorrowDate < '2006/05/25')
          JOIN
          (SELECT * FROM Item I WHERE Author = 'JoePblic')
          ON I.ItemId = B.ItemId
          )
)ON  C.CustId = B.CustId ));
```

Answer 3:

The main query selects the unique title with a specific code C.code.

This code is determined by the result of inner subquery

Inner Subquery →
It groups enrollments by course (C1.code) and counts how many students are registered in each course, regardless of department.

It returns code of courses maximum member of "cross-department" enrollments as determined by nested subquery.

Nested Subquery →
It calculates the maximum count of cross-department enrollments for any course, means that the course being taken by students of different department than the one offering the course (c2.dept <> s2.dept)

Final →
The query finally retrieves the title of the course that has the maximum number of students from departments other than offering the course.

Answer 4:

Given two transactions & schedules as :-

T_0	T_1
$r_0[A]$	
$w_0[A]$	
	$r_1[A]$
	$r_1[B]$
$r_0[B]$	C_1
$w_0[B]$	
C_0	

Identifying Conflict:

Conflict between $w_0[A]$ and $r_1[A]$

$w_0[A]$ (write by T_0 on A) and $r_1[A]$ (read by T_1 on A) conflicts because they operates on same data item A & one is write & another One is read.

∴ we have a conflict between T_0 and T_1 on A.

Conflict between $w_0[B]$ and $r_1[B]$:

$w_0[B]$ (write by T_0 on B) and $r_1[B]$ (read by T_1 on B) conflict because they operate on same data item B and one is a write and the other is read.

∴ we have a conflict between T_0 & T_1 on B.

Now, based on the conflict identified:

(i) from the conflict between $w_0[A]$ (by T_0) and $r_1[A]$ (by T_1): T_0 must come before T_1, so we add a directed edge from $T_0 \rightarrow T_1$ in the precedence graph

(ii) from the conflict between $w_0[B]$ (by T_0) and $r_1[B]$ (by T_1): T_0 must come before T_1, so we add another directed edge from $T_0 \rightarrow T_1$. Therefore the precedence graph will look as :-

$T_0 \rightarrow T_1$

Now, since precedence graph does not contains any cycles, so the given schedule is conflict serializable.

Answer 5:

Transaction T_1 :

```
lock - S(A)
 read (A)
lock - S(B)
 read (B)
if A=0 then
    upgrade to lock - X(B)
      B := B+1
      write (B)
    unlock (B)
    unlock (A)
```

Transaction T_2 :

```
lock - S(B)
 read (B)
lock - S(A)
 read (A)
if B=0 then
    upgrade to lock - X(A)
      A: A+1
      write (A)
unlock (A)
unlock (B)
```

Deadlock Possibility :- Deadlock will occur if - T_1 locks A and waits for lock B, while T_2 locks B and waits to lock A.
This is called circular wait and is a classic deadlock condition. If both transaction acquire their initial locks but then wait on each other's locked resources, neither can proceed, resulting in a deadlock.