CS3700 - Introduction to Database Systems
End Semester Examination - 21st - 22nd Dec

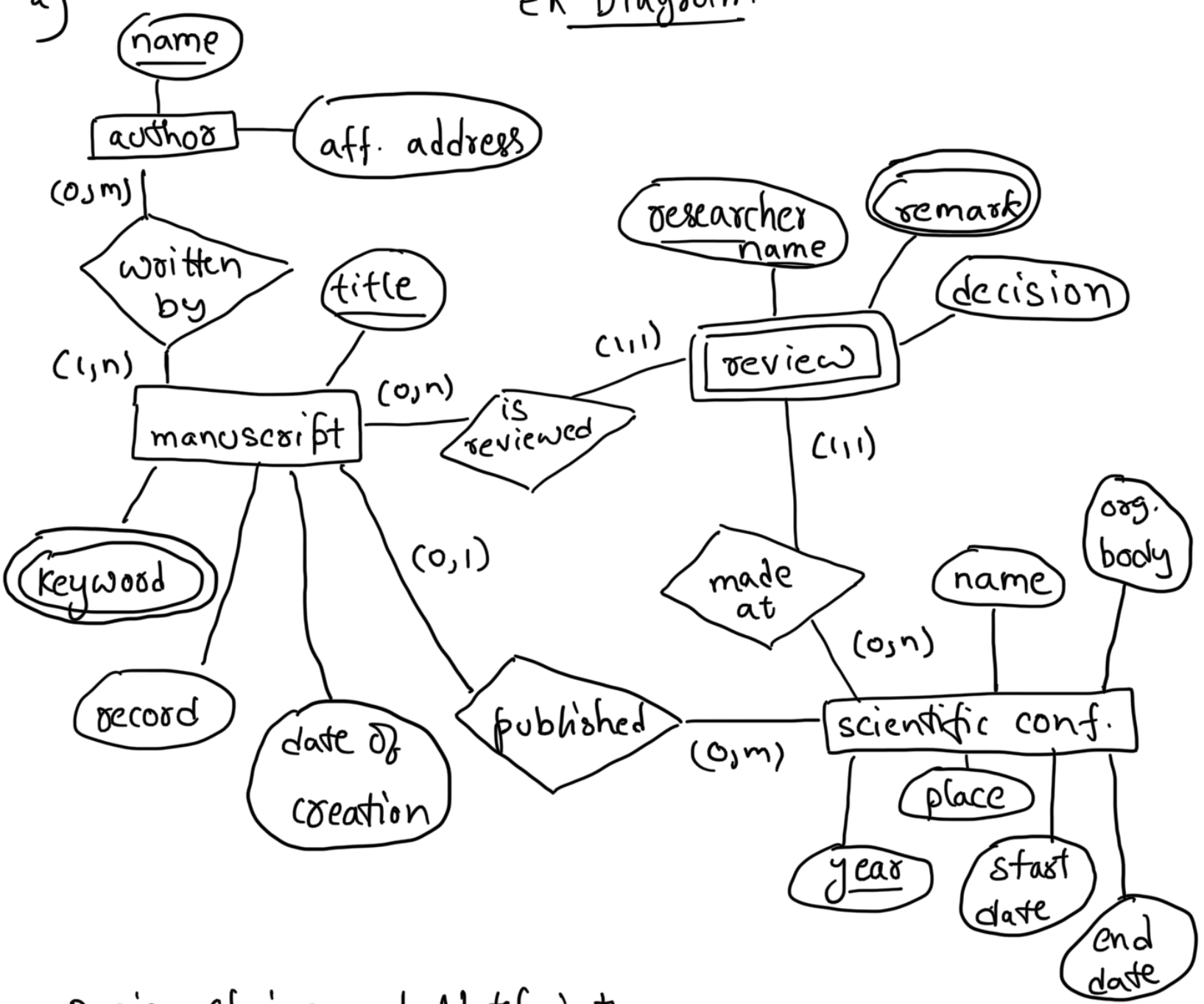
I understand that take-home examination is to test my own understanding of the subject and I will offer an honest attempt to meet this goal. I will not communicate with anyone else while working, answering the examination problems / tasks. I will not post the questions to any public fora and seek answers. I will not help other course mates taking the examination in any manner I understand that I can consult books, course slides and videos by the teacher while answering the examination. I also understand that I should not copy materials from books, internet sites etc as a part of my answers for the examination questions / tasks. These restrictions apply for whole examination duration (till the deadline for submission).

ARABHI SUBHASH
CS17B005

Task #1

Smod3 = 2 ; 2+1 = 3 ; working on 3rd domain description.

a) ER-Diagram



Design choices and Notepoints

1) Didn't maintain separate relation for "submitted" because that info. can be obtained by checking whether there is a review for that manuscript at that conference (Implicit)
2) Made review a weak entity whose existence depends on sci. conf. and manuscript. The reason for this is to accomdate multiple reviews

for a pair of conf. & manuscript

3) Assumed title is unique for manuscripts or else we can also add an attribute named man. id. and make it a primary key

4) Maintained a relation "published" with min-max constrain (0,1) to impose restriction that a manuscript can be published at exactly one conference or not published yet

5) keyword, remark are multivalued attributes

6) The contents of book "proceedings" of a certain conference is implicitly stored in the relation "published"

b) Relation Schema Diagram

" ⟶ " - foreign key

" --- " - primary key

Task #2

a)

Query: Get the courseId and name of courses which are all male students in odd sem of 2002. Assume every course is offered and has atleast 1 student

enrolled in the given semester and year

Query TRC : $\{ C.courseId, C.name \mid course(C) \wedge (\forall s)((student(s) \wedge enrollment(e) \wedge e.year = 2002 \wedge e.courseId = C.courseId \wedge e.sem = 'odd' \wedge s.rollNO = e.rollNO) \rightarrow (s.sex = 'male')) \}$

Explanation: Basically we are picking all the students enrolled in the course and checking the genders. If all are male then $\forall$ condition satisfies and outputs that tuple

Query Relational Algebra:

1) courses-with-female $\leftarrow \Pi$ courseId (σ sex = 'female' (Π rollNO, courseId(σ year = 2002 & sem = 'odd' (enrollment)) $\bowtie$ Π rollNO, sex (student)));

2) courses-only-male $\leftarrow$ (Π courseId(course) − (courses-with-female);

3) result $\leftarrow \Pi$ courseId, name (course $\bowtie$ courses-only-male);

Explanation: In part 1 collected all the courses in given sem and year with female students. Did that using inner join of student & enrollment. In second part deleted them from all courses to get male only courses. In last part joined it with courses to get course name

into output.

b) query : Get the name and rollNo. of all the students who did a 4 credit course in even semester of 2002

Sql query : select s.name, S.rollNo from students s, enrollment e, course c where e.rollNO = S.rollNo and e.courseId = C.courseId and e.sem = 'even' and e.year = 2002 and C.credits = 4

Relational Algebra expression :

$\Pi_{s.name, S.rollNO}(\sigma_{\alpha}(S \times e \times c))$ where s: student e: enrollment, C: coorse

α = (e.rollNO = S.rollNo and e.courseId = C.courseId and e.sem = 'even' and e.year = 2002 and C.credits = 4)

optimisations using main heuristic role :

1) Π S.name, S.rollNO $(\sigma_{B_1}(S \times \sigma_{p2}(e) \times \sigma_{p3}(C))$

B_3 = (C.credits = 4)

B_2 = (e.sem = 'even' and e.year = 2002)

B_1 = (e.rollNO = S.rollNo and e.courseId = L.courseId)

→ pushed independant constrains inside of brackets

2) Π S.name, S.rollNO. $(\sigma_{\theta_1}(S \times \sigma_{\theta_2}(\sigma_{\theta_3}(e) \times \sigma_{\theta_4}(C))$

θ_4 = (C.credits = 4)

θ_3 = (e.sem = 'even' and e.year = 2002)

θ_2 = (e.courseId = C.courseId)

θ_1 = (e.rollNO = S.rollNO)

→ further pushing of constrains

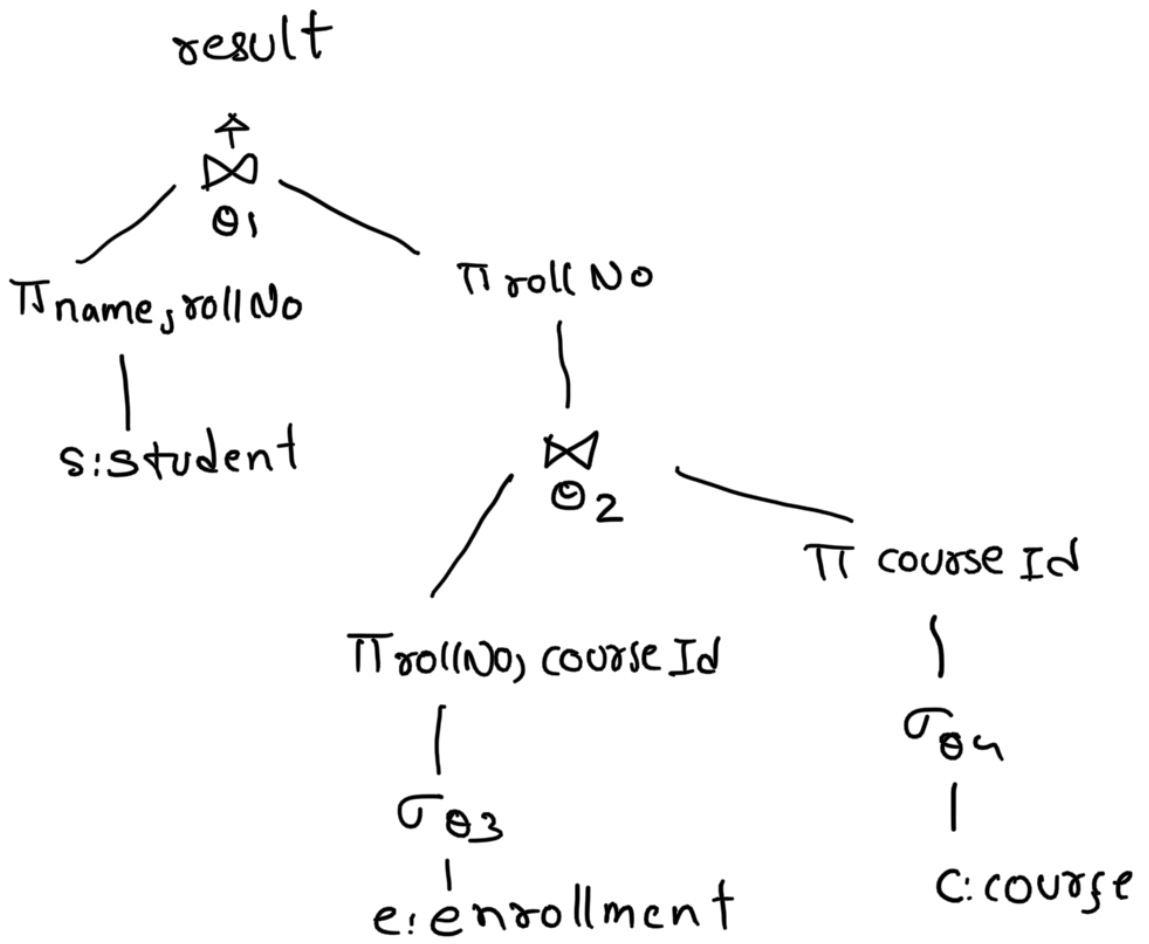
3) $\Pi_{s.name, s.rollNo}(S \bowtie_{\theta_1} (\sigma_{\theta_3(e)} \bowtie_{\theta_2} \sigma_{\theta_4(c)}))$

→ converted cross product + selection into join.

4) $(\Pi_{name, rollNo}(S) \bowtie_{\theta_1} \Pi_{rollNo}(\Pi_{rollNo, courseId}(\sigma_{\theta_3(e)}) \bowtie_{\theta_2} \Pi_{courseId}(\sigma_{\theta_4(c)})))$

→ Finally, Applied projection to lower levels

Above RA expressions with brackets is equivalent to tree representation. Final expression tree is



c) query: Get max and min & avg credits offered by each department. Assume each dept. offers atleast one course.

SQL query: create view avg_credits_dept as
select c.deptNo, avg(c.credits) as avg_credits
from course c group by c.deptNo;
select max(avg_credits), min(avg_credits) from avg_credits_dept.

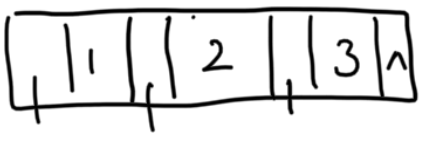
Explanation: In Given query we need to max & min & avg (aggregate over aggregate) so we need views.

In the first part using avg we have created a view avg-credits_dept which stores deptNo. and its avg. credits. Finally we applied min-max on that view.

Task #3

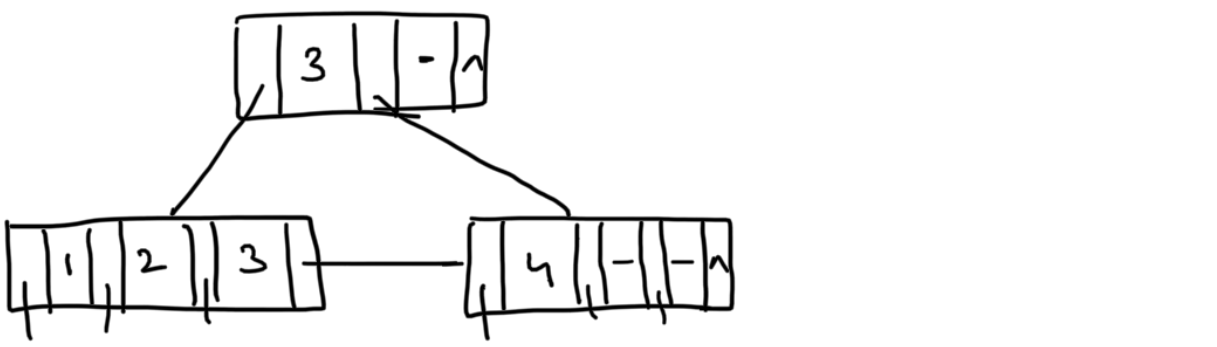
a) $0 = 3$ $0_{leaf} = 3$

i) Insert 1,2,3

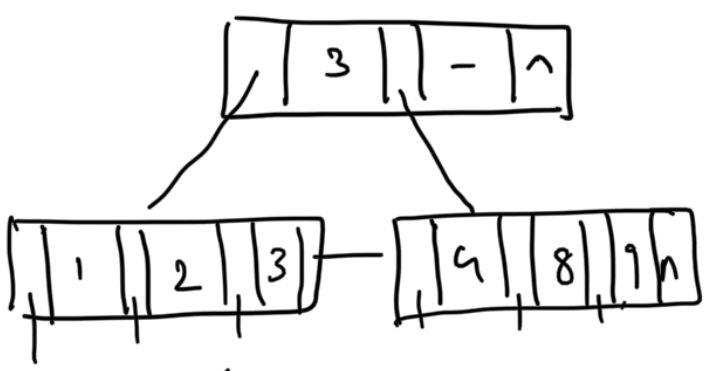


ii) Insert 4

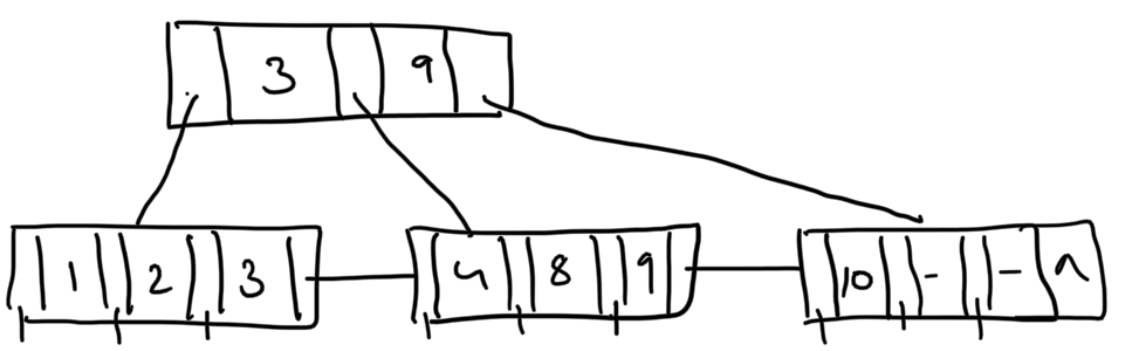
overflow occurs at $\lceil \frac{3+1}{2} \rceil = 2$, leaf node



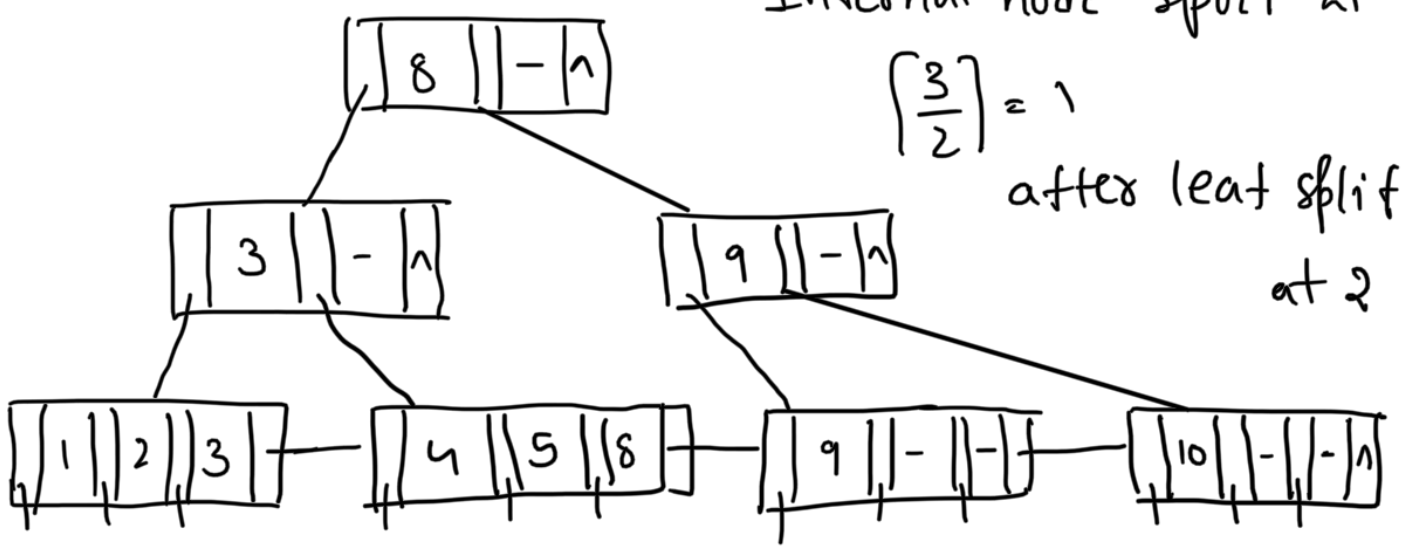
iii) Insert 8, 9



iv) Insert 10 Same, overflow at 2



v) Insert 5



After 8 inserts we saw 2 leaf splits and 1 internal node + leaf split. the final tree above still have spaces but we stopped inserting as we have seen the 2 different splits and reached 2 levels.

b) pseudo code for linear hashing insert:-

```
M = 1 # initial length of bucket
N = 0 # split pointer
i = 0 # present set of hash functions

def hash_func(i,x):
    return x%(pow(2,i)*M)

def split_bucket():
    # add bucket (pow(2,i)*M)+N to the set of buckets
    for values: v in bucket N and it's over_flow:
        if hash_func(i+1,v) != N:
            # remove v from bucket N
            # add v to bucket (pow(2,i)*M)+N (Newly added Bucket)

def place_record(b,x):
    # check if bucket b is full
    if not full:
        # add x to bucket b
    else:
        # add x to over flow of bucket b
        split_bucket()
        if N == (pow(2,i)*M)-1:
            i+=1
            N = 0
        else:
            N+=1

def insert(x):
    if hash_func(i,x) < N:
        place_record(hash_func(i+1,x),x)
    else:
        place_record(hash_func(i,x),x)
```

Brief: Above is pseudo code for linear hashing where hashing functions are $(x \bmod 2^i \cdot M)$. Simple adding to bucket and adding to overflow are presented as comments

Task #5

a) Given statment has if and only if (double implication) so proof will be in 2 parts

1) forward implication

→ if a schedule is conflict serializable then precedence graph G has no cycles

proof by contradiction:

→ consider a schedule X which is conflict serializable and has a cycle in G $\equiv A \wedge \sim B \equiv \sim(\sim A \vee B) \equiv \sim(A \rightarrow B)$

→ let the cycle be $T_1, T_2, T_3 \ldots T_k$

→ this implies that there must be operations $OP_1(x)$ $OP_2(x)$ $\ldots$ $OP_k(x)$ $OP_1(x)$ in the schedule

→ To make it conflict serializable we must bring the first and last OP_1 operations together - There are 2 ways of doing this

→ one way is to move the first operation forward but this is not possible due to conflict with $OP_2, OP_3 \ldots OP_k$ of lower precedence

→ other way is to move last op_i back but that too conflicts due to higher precedence of same set of operations

→ Hence we can say that our assumption is false and by contradiction the required forward implication is true.

2) Backward implication

→ If precedence graph G is acyclic then schedule is conflict serializable.

proof by induction on number of Transactions:

Basis step: n=1

→ It it trivally true for one transaction

Hypothesis step: n = k

→ Assume that for k transactions the reqd. implication is true

Induction step: n = k+1

→ Given precedenc. graph is acyclic then there will be atleast one node with no in-edge. let this node be Tm

→ As there is nothing preceding transaction m, all the operations corresponding to this transac-tion can be swapped to start without any conflict

$OP_m(x)$ -- $OP_1(x)$ -- $OP_m(x)$ -- $OP_2(x)$ .. $OP_m(x)$

transformed to

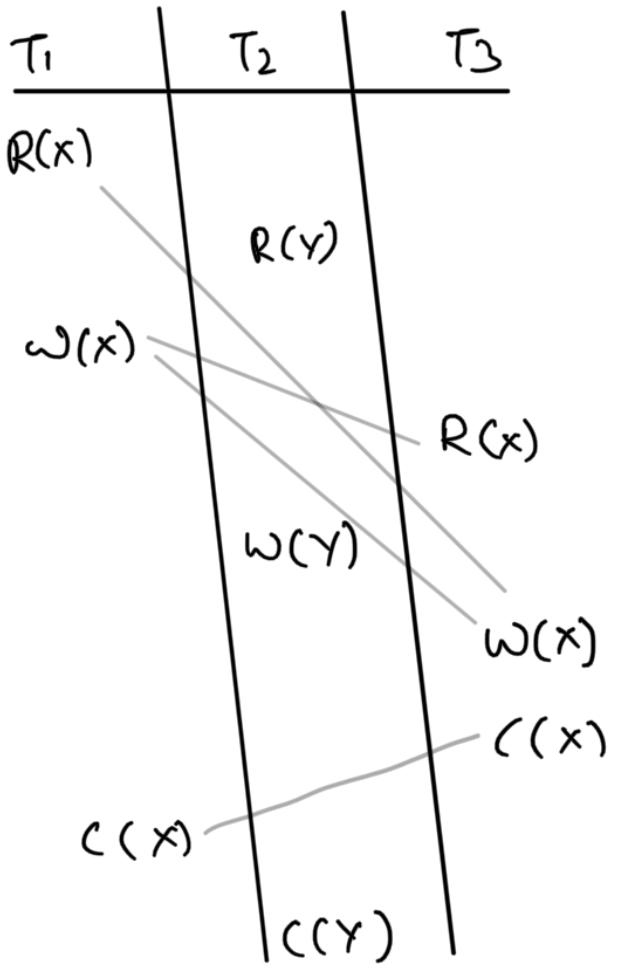
$OP_m(x)$ $OP_m(x)$ $OP_m(x)$ --- $OP_1(x)$ $OP_2(x)$ ---

→ So the original schedule S can be written as $S = T_m ; S'$. here S' is the schedule of remaining K transactions

→ From hypothesis step we can say that S' is conflict serializable. By merging this with transaction m we can say that our assumption is true for n = k+1

→ From induction we can conclude that our backward implication is true for any number of transactions

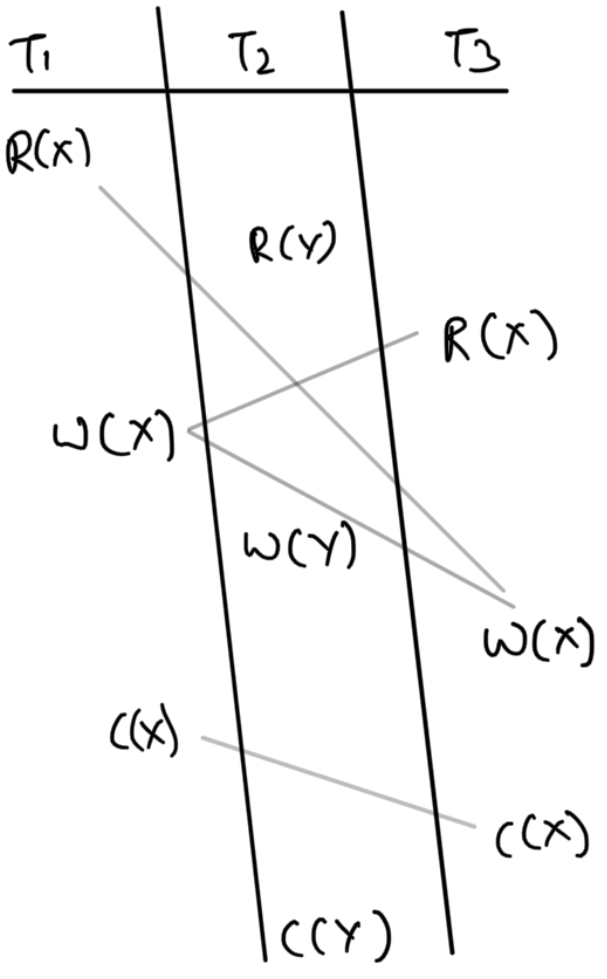
b) i) non recoverable
conflict serializable



T3 which reads from the written value of T1 commits before it

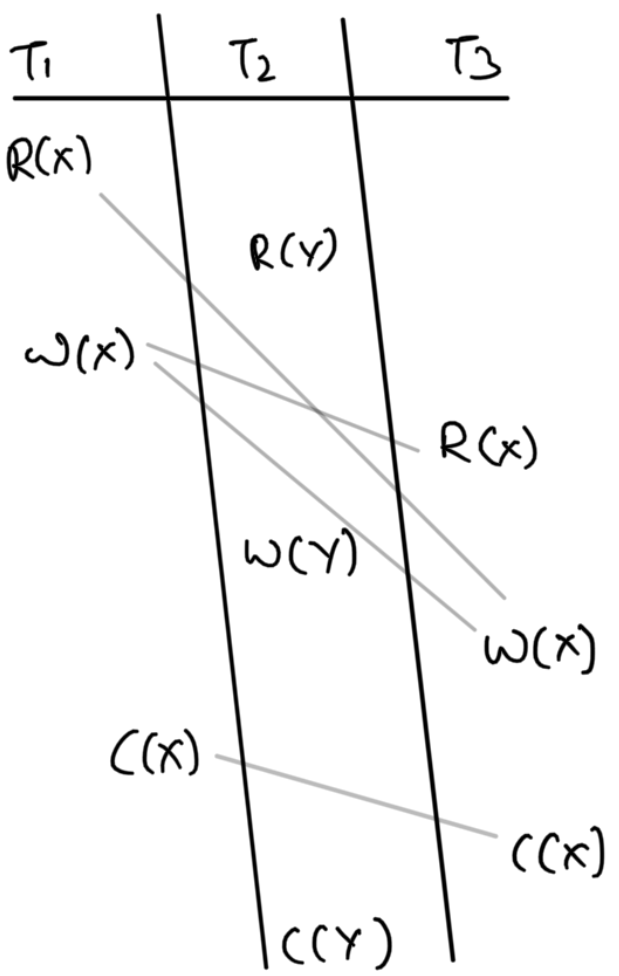
→ C(X) → commit X

ii) recoverable
not conflict serializable



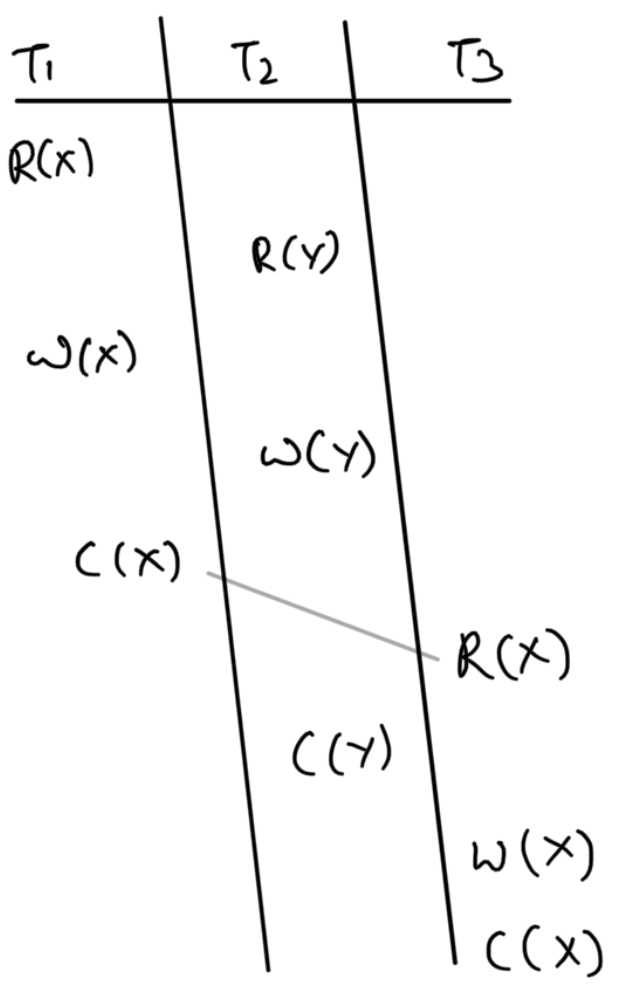
T3 reads before T1 writes - conflict

3) recoverable
conflict serializable
not cascadeless



T3 read from T1 after it writes and before it commits - cascading
- no conflicts

4) recoverable
conflict serializable
cascadeless



T3 reads after T1 commits it's written value
- no cascading
- no conflicts too

c) REDO logging stores the final values of the transaction. As transactions proceed in forward direction so should the reading of log be. Below is a small example to show it

Consider a state where A has 500/- and B has 1000/-. From this state 2 transactions going to happen - i) A spends 100/-

ii) A spends 2001-

State Table

Step	Action	m	Mem-A	Disk-A	Log
1)					<start T_1>
2)	$R_1(A,m)$	500	500	500	
3)	$m = m-100$	400	500	500	
4)	$W_1(A,m)$	400	400	500	<T_1, A, 400>
5)					<commit, T_1>
6)	Flush log				
7)	output(A)	400	400	400	
8)					<start, T_2>
9)	$R_2(A,m)$	400	400	400	
10)	$m = m-200$	200	400	400	
11)	$W_2(A,m)$	200	200	400	<T_2, A, 200>
12)					<commit, T_2>
13)	Flush log				
14)	output(A)	200	200	200	

→ If crash occurs before 6 then commit isn't present in the disk and the values are also not updated so no action required.

→ If it occurs after 6 and before 13 Transaction T_1 is redone using log entries and no action required for T_2

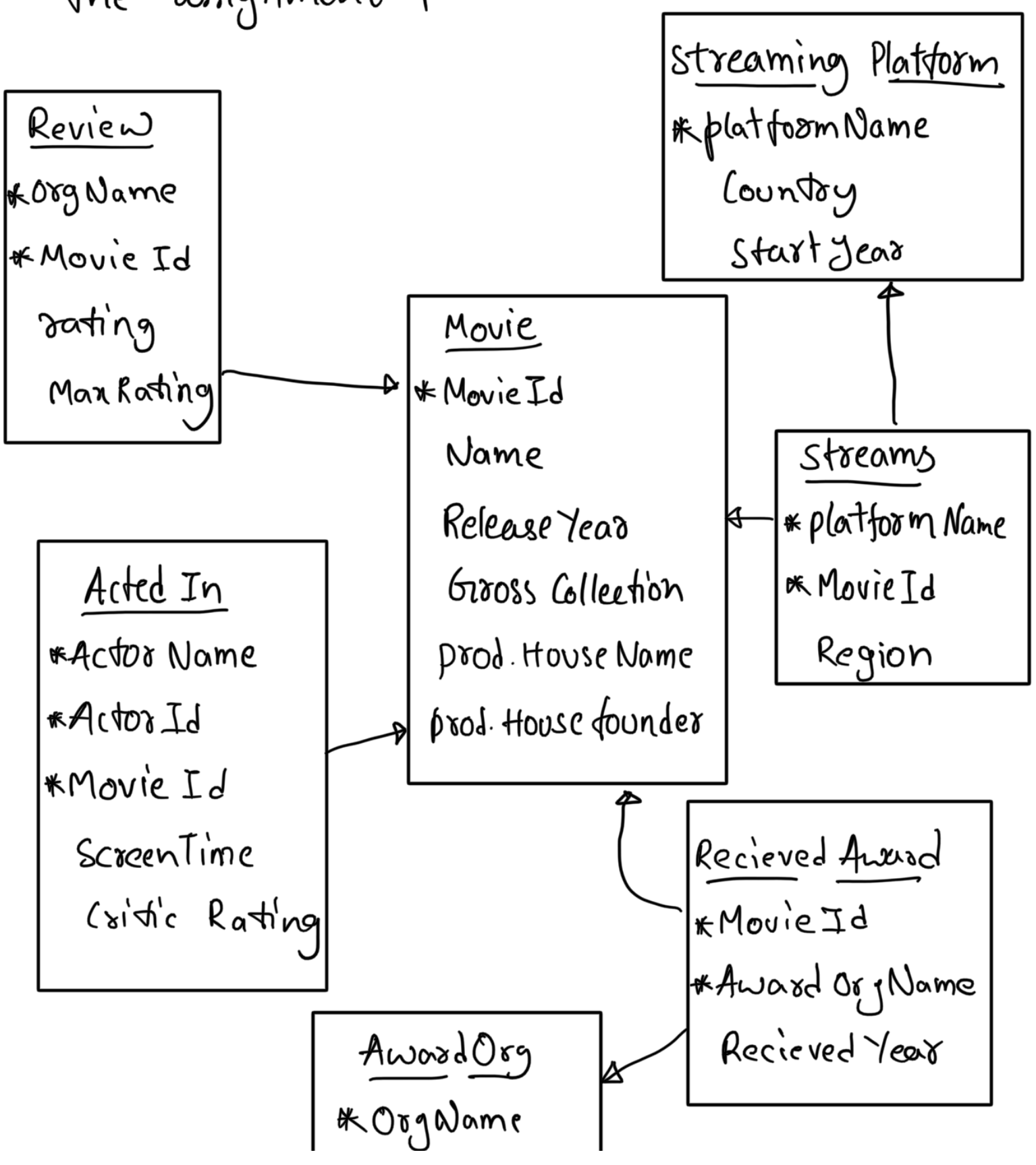
→ Consider crash occurs after 13 and we don't know whether result of T_1 or T_2 flushed to disk. Here we need redo transactions from start as we have both

commits in disk. Forward reading of log puts a value of 200 to the disk-A where as backward or other readings will write 400.

we can see from the table that former (forward) reading is only correct as transactions happen in this direction.

Task #4

a) Modified DB scheme of database designed in the assignment-1



Country
Start Year

* → key

→ Relations are modified st. u different Normal forms can be observed

→ Name of attribute intuitively provides its meaning. Few of them are explicitly defined below

MovieId :- Unique Id given to a movie

orgName, PlatformName :- unique name given to organization or platform respectively

StartYear :- Established Year of the company

streams: Region :- The region/part of world where the movie can be streamed by the platform.

ActorId :- unique Id given to each actor in the movie industry

Actor Name :- Name of Actor (Assume it's unique)

MaxRating :- The scale on which rating is given starting from 0 by Revieworg.

Ex: 10 → (0-10), 100 → (0-100)

prod. House :- production House

Screen Time :- The total time actor is present on screen during the movie

Critic Rating :- Avg performance rating given by critics

→ Functional Dependancies in each relations and

their highest normal form are described below

1) Review key - (orgName, MovieId)

FD - (orgName, MovieId) → Rating

* Each org. gives a single rating to movie (overall rating)

- orgName → MaxRating

* Each org. has specific scale on which they rate movies

NF - Not 2NF as non prime attribute (MaxRating) is not fully functionally dependant on key (orgname, MovieId) So it's just 1NF

2) Movie Key - MovieId

FD - MovieId → Name, ReleaseYear, Grooss
prod. House Name
prod. House founder

* MovieId is uniquely given to a movie

- prod. House Name → prod. House founder

* Assuming naming is unique and a house has only one founder

NF - clearly it's 2NF as every attribute depends on single key (MovieId) but it's not 3NF as there is a transitive relation -

MovieId → prod. House Name → prod. House founder

3) Acted In keys - (ActorId, MovieId),
(Actor Name, MovieId)

FD- (ActorId, MovieId) → Screentime, Critic rating

- (ActorName, MovieId) → Screentime, Critic rating

* As actor can't have two or more screen times or avg critic ratings for a movie.

- ActorId → ActorName
- ActorName → ActorId

* Assuming name and Id are unique for an actor in movie industry

NF - Clearly its 3NF as its 2NF and no non prime transitive property but Its not BCNF as ActorId → ActorName is nontrivial FD and ActorId is not a superkey.

4) Streaming platform key - platformName

FD- platformName → Country, StartYear

* trivial, Name is unique for a platform

NF - BCNF, In every nontrivial FD X → A we can see X is superkey

5) Streams key - (platformName, MovieId)

FD- platformName, MovieId → Region

* A company buys streaming rights of a movie for a region

NF - BCNF

6) Award Org key - OrgName

FD- OrgName → Country, StartYear

* trivial, Naming is unique

NF - BCNF

7) Recieved Award Key - (MovieId, AwardOrgName)

FD - (MovieId, AwardOrgName) → RecievedYear

* Trivial, A film is considered for awards by an org. only one time (one year).

NF - BCNF

b) Given : If the matrix operated on by L-Test doesn't have a row of all "a" symbols at the end of it's running then the given decomposition is lossy wrt given FDs

Proof:

Consider a relation Schema $R = A_1, A_2 \dots A_n$ (A_i - attribute) and the decomposition $S - R \to R_1, R_2, R_3 \dots R_k$

L-Test : A table of k×n is constructed with R as rows and A as columns. This table is instantiated with b_{ij}.

→ Now if $R_i \to A_j \dots A_k$ then the values $b_{ij} \dots b_{mk}$ are replaced with $a_j \dots a_k$

⇔ This step is equivalent to adding an instance $a_1 a_2 \dots a_n$ to R (adding a data point $(a_1 a_2 \dots a_n)$ to initial Relation Schema)

→ After that we use FDS to change the values of matrix

⇒ let us consider an FD $A_j \to A_k$ and R_i has elements a_j and b_{ik}.

As functional dependancy independant of decomposition from R we can say if A_j is a_j then A_k is a_k. Therefore we can change the value of b_{ik} to a_k

→ Even After robustly applying FDs if we couldn't get a row with all "a"s means that our decomposition doesn't have the instance $a_1 a_2 \ldots a_n$ wrt given FDs

→ As the instance $a_1 a_2 \ldots a_n$ present in R lost after decomposition, this is a lossy decomposition wrt given FDs