Thapar Institute of Engineering & Technology, Patiala

Computer Science and Engineering Department

END-SEMESTER EXAMINATION (Solution)

Course Code: UCS310	Course Name: Database Management Systems
B.E. (Second Year), Semester-II	Branch: COE, CSE
May 15, 2024	Wednesday, 9.00 AM – 12 Noon
Time: 3 Hours, M. Marks: 40	Name of Faculty: RKR, MK, RRN, SKH, CHP, SRO, AAD, VAP

Q1. a) Q1. (a) Consider a relation R (M, N, O, P, Q, R). Find the minimal cover of the following functional dependencies holds in R. Show clearly all the steps while finding the minimal cover. **2M**

M→NO, OP→Q, Q→O, P→MQR, MNR→NP, PR→NO

Q1. (b) Consider a relation *EMP(emp_id, dept_id, ename, mgr_id, age)* where the following functional dependencies hold. (emp_id, dept_id) →ename, emp_id→mgr_id, emp_id→age, mgr_id→age. (Assume all the attributes having atomic values.)

(i) State whether an update anomaly can occur in the relation EMP or not with *proper reason*. **2M**

(ii) State in which normal form is EMP now by identifying the prime and non-prime attributes. Convert it to higher normal forms till BCNF, stating that *lossless join is satisfied* at each normal form. **4M**

(iii) Does multi-value dependency (MVD) exist in the following relation R (A, B, C)? If YES, then justify why such MVDs exist and list them all. If NO, then justify why not exist in R. **2M**

A	B	C
3	3	3
1	1	1
1	1	2
1	2	1
1	2	2
2	2	1
2	2	2
4	4	4
3	3	1

Answer: 1. a) Minimal Cover holds in R: {M→NO, Q→O, P→MQR, MR→P}.

1 mark for reasonable intermediate steps.

Full 2 marks for intermediate steps and correct minimal cover.

1. **b) i)** *Yes*, update anomaly exists in the relation EMP.

Reason: This is due to emp_id→mgr_id. This shows one-one or many-one relationship exist between emp_id and mgr_id. Hence if the manager changes then in many places of the relation EMP, for the same value of emp_id, we need to change the mgr_id. [2 marks if reason is correct]

ii) (emp_id, dept_id)$^+$ = emp_id, dept_id, ename, mgr_id, age. Hence, (emp_id, dept_id) is the composite primary key. **The prime attributes are emp_id and dept_id**. **Non-prime attributes are ename, mgr_id, age.**

The relation EMP is in first normal form as all the attributes having atomic values. [1 mark]

Conversion to 2NF: Partial dependency exist in the relation. emp_id→mgr_id and emp_id→age. Hence it is not in 2NF. So, we decompose the relation into two sub-relations R1 and R2 as follows:

R1:(emp_id, dept_id, ename), FD: (emp_id, dept_id) →name

R2: (emp_id, mgr_id, age), FDs: emp_id→mgr_id, emp_id→ge, mgr_id→ge

Lossless join satisfied here. The common column between R1 and R2 is emp_id and it is a primary key in R2. [1 mark]

Conversion to 3NF: In R1, there is no transitive dependency, but in R2, transitive dependency exists. This is due to: emp_id→mgr_id, mgr_id→age. Hence R2 is not in 3NF. So, we further decompose R2 into two sub-relations to avoid transitive dependency.

R21: (emp_id, mgr_id), FD: emp_id→mgr_id.

R22: (mgr_id, age), FD: mgr_id→age.

Lossless join satisfied here. The common column between R21 and R22 is mgr_id and it is a primary key in R22. [1 mark]

Conversion to BCNF: All the sub-relations of EMP R1, R21, and R22 are in BCNF, since the left-hand side of each FD is a candidate key. [1 mark]

(iii) The MVDs exist in the relation R are: A→→B, A→→C, B→→A, B→→C.

Generic reason: For any two attributes (Say A and B), A→→B exist in R, iff we have any two tuples t1 and t2 in R with the same A value and if we exchange the B values of these tuples and called the tuples so obtained as t3 and t4, then the tuples t3 and t4 must also be in R. [0.5 mark for each such MVD. No marks for huge random list of trivial/non-trivial MVDs]

Q2. (a) Consider the schedule S1 given below having three transactions T1, T2, and T3.
S1: R2(A), R1(B), W2(A), W1(B), C1, R3(A), W3(A), R2(B), W2(B), C2, C3.
(i) Show that whether the schedule S1 satisfying 2PL by adding lock and unlock instructions to T1, T2 and T3.

Marks: 1.5 Marks- Schedule with correct Lock/unlock sequence. 0.5 for justification

Answer: Schedule S after applying the 2-Phase Locking i.e. applying Lock and Unlock of data items using the rule of 2-PL is given below. In the 2-Phase locking protocol, a transaction can acquire a lock until it has not performed any unlock operations.

2PL Using Shared, Exclusive And Unlock operations	**T1**	**T2**	**T3**
		X(A) R(A)	
	X(B) R(B)		
		W(A)	
	W(B) Commit **U(B)**	**X(B)** **U(A)**	
			X(A) R(A) W(A)
		R(B) W(B) Commit **U(B)**	
			Commit **U(A)**

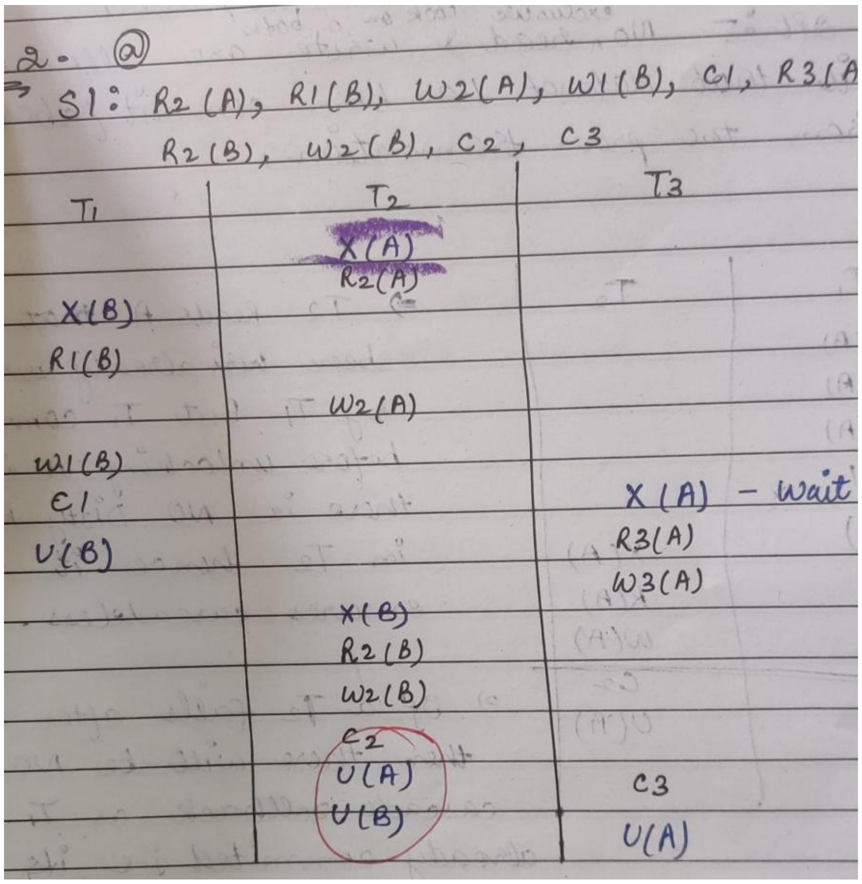
As we can confirm form the schedule after applying, the Lock and Unlock operations in growing and shrinking phase order, 2PL is applicable in the schedule.

OR

1 mark is given for following schedule

2PL Using Lock And unlock operations	**T1**	**T2**	**T3**
		L(A) R(A)	
	L(B) R(B)		
		W(A)	
	W(B) Commit **U(B)**	**L(B)** **U(A)**	
			L(A) R(A) W(A)
		R(B) W(B) Commit **U(B)**	
			Commit **U(A)**

Reason is same as mentioned above. The schedule after applying, the Lock and Unlock operations in growing and shrinking phase order, 2PL is applicable in the schedule.



2. ⓐ

→ S1: R2(A), R1(B), W2(A), W1(B), C1, R3(A

R2(B), W2(B), C2, C3

T1	T2	T3
	~~X(A)~~ ~~R2(A)~~	
X(B) R1(B)		
	W2(A)	
W1(B) C1 U(B)		
		X(A) – wait R3(A) W3(A)
	X(B) R2(B) W2(B) C2 U(A) U(B)	
		C3 U(A)

2(a) (ii) Explain with examples how Strict and Rigorous 2-PL ensures cascadeless but not able to handle deadlock. Check whether schedule S1 is satisfying the strict-2PL locking protocol or not with proper justification.

Marks:

Answer: Consider the schedule **P** given below, we can see that 2PL is applicable but it is not free from cascading. If the transaction T1 fails just before its commit point, both transactions T2 and T2 will be forced to rollback as this transaction has performed read operation on the data item A which is modified by T1.

P	**T1**	**T2**	**T3**	

	X(A)
	R(A)
	W(A)
	S(B)
	U(A)
	R(B)			

		X(A)
		R(A)
		W(A)
		U(A)		

			S(A)
			R(A)	

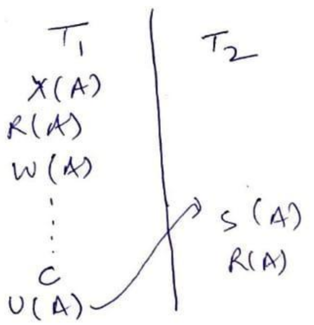
	Failed point **Commit**			
	U(B)	Commi t		
			Commi t U(A)	

In S1, T1 is forced to unlock X(A) before commit so it is not satisfying the Strict-2PL/ Rigorous 2-PL locking protocol.

Strict-2PL: In the case of strict 2-PL a transaction should hold it should be in 2PL and all exlusive locks should be hold until commit of the transaction. 2-PL is Recoverable and Cascadeless, but not free from Deadlock.

Now, consider the schedule **Q** given below:

Example



In this Example T1 wants to perform read and write on A so, it will demand exclusive lock. T2 wants to perform read on A So, it will wait untill exclusive lock is unlocked by T1. As per Strict 2PL T1 will only unlock the exclusive lock if Transaction commits / rollbacks. In this case the changes will permanently stored in the database and ~~the~~ then T2 will read from the database. So, The limitation of Cascading roll back will be removed.

So, strick 2PL will always produce Cascadeless schedules. In Regorous 2PL even shared locks are held until commit So, it will also ensure Cascadeless Schedules

S1 Schedule after applying locks as per strict-2PL

After Using Shared, Exclusive Locks and Unlock operations	**T1**	**T2**	**T3**	As we can see here T2 is forced to unlock the exclusive lock X(A) before its commit point. Therefore, this schedule is not satisfying the Strict-2PL protocol.

		X(A)
		R(A)		

	X(B)
	R(B)			

		W(A)		

	W(B)
	Commit
	U(B)
		X(B)
		U(A)		

			X(A)
			R(A)
			W(A)	

		R(B)
		W(B)
		Commit
		U(B)		

			Commit **U(A)**	

Q2. (b) Consider the schedule S2 given below having five transactions T1, T2, T3, T4 and T5.
S2: R1(A), W1(A), R3(A), W3(A), R5(A), R4(B), W4(A), W2(A), C1, C2, C3, C4, C5.
Suppose T1 transaction fails just before its commit (*C1*) point, what are the other transactions which will be forced to be rollback, and explain why such rollback happened? **2M**

Mark: 1- mark for showing the schedule and correct answer and 1 Mark for correct justification.

T1	**T2**	**T3**	**T4**	**T5**	

R(A)
W(A)
					If transaction T1 fails at the mentioned failing point (i.e. just before its commit), then only T3 and T5 will be forced to rollback but not T2 and T4.

		R(A)
		W(A)			

				R(A)	T3 and T5 will be forced to rollback as it has performed read operation on the data items A which is modified by the uncommitted transaction T1.

			R(B)
			W(A)		

	W(A)				
Failing Point					T2 and T4 will not require rollback as these transaction has not performed any read operation on the data item A which has been modified by T1.
Commit	Commit	Commit	Commit	Commit	

Q3. (a) Consider the following schedule S3 having four transactions T1, T2, T3, and T4.

S3: R1(A), W3(A), W3(B), R2(A), W1(A), W1(B), W2(A), W2(B), W4(A), W4(B)

(i) State whether S3 is conflict serializable or not by using the precedence graph.

(ii) Check for the view serializability by explaining the view serializability conditions. List all the equivalent serial schedules.

Marks: (i) 1.5: Drawing correct precedency graph and identifying the cycle. 0.5: explanation for not conflict serializability.

(ii) 0.5: Identification of Blind write. 1: showing and explaining conditions view-serializablity and 0.5: for listing the equivalent serial schedule.

Answer: For simplicity and better understanding, we can represent the given schedule pictorially as-

S3	**T1**	**T2**	**T3**	**T4**
	R(A)			

			W(A)
			W(B)	

		R(A)		

	W(A)
	W(B)			

		W(A)
		W(B)		

				W(A)
				W(B)

We know, if a schedule is conflict serializable, then it is surely view serializable. So, let us check whether the given schedule is conflict serializable or not. To Checking Whether S3 is Conflict Serializable or Not, Draw the precedence graph using by examining the conflicting operations.

Precedence graph for the schedule S3:

As we can see that the graph consists of cycles, so the schedule **S3 is non-conflict serializable** but the schedule may or may not be serializable.

Now, we need to identify if there is any **blind write** present in the schedule?

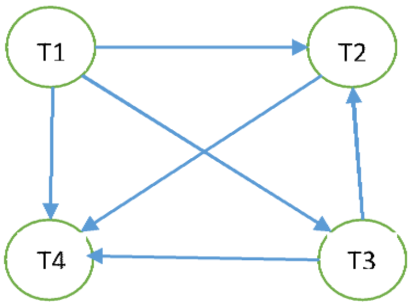
Yes! Following blind write present in S3.

T1: W(B), T2: W(B), T3: W(A) and W(B), and T4: W(A) and W(B)

This means that schedule may or may not be serializability. For this we need to check for the view serializability conditions.

View Serializability Conditions:

- **A/c to Initial Read conditions:** T1 performs the initial read on the data item A followed by R(A) of T2. So T1 should be executed be before T2. Possible Schedule: T1→{T2, T3, T4}
- **As per final write:** T4 performing the final write on both data items A and B so T4 should also perform the final write in view equivalent serial schedule. Therefore, the possible schedule will be: T1→{T2, T3}→T4
- **Finally, as per Updated Read rule:**
    - T2 is reading the data item A, R(A), which is modified by T1: W(A). It means there is no updated read violation in the case of T1→T2.
    - Now, we can see T2 is reading the data item T3: W(A). Which means T3 should be executed before T2 to not violate the updated read condition. i.e. possible schedule is T3→T2.
- **<u>Dependency Graph using the view serializability conditions:</u>**



Using topological sequence, the view serial equivalent schedule of S3 will be:

T1→T3→T2→T4

Q3. b) (i) Consider the schedule S4 given below having three transaction T1, T2, and T3.
S4: R1(X), R3(Z), R2(Y), W2(Y), W1(X), W3(Z), W1(Y), C1, R3(X), C2, C3.
Check whether the schedule S4 is cascadeless recoverable or not with proper justification. Convert the schedule to strict recoverable schedule by changing the order of commit operations (Note: do not change the order of the read and write operations).
(ii) Explain with suitable example why blind writes are not allowed in strict cascadeless schedule.

Marking: 2 for checking cascadeless recoverable schedule (Schedule Diagram+justification), 2 marks (trict recoverable schedule+Justification)

S4:	**T1**	**T2**	**T3**
	R(X)		
			R(Z)

		R(Y)
		W(Y)	

	W(X)		
			W(Z)
	W(Y) Commit		
			R(X)
		Commit	
			Commit

Cascadeless recoverable schedule: A schedule is cascadeless recoverable if any Read operation on a data item A performed by transaction T which is already modified by transaction T' then R(A) of T should be done after commit of T'.

As we can see in the schedule S4, R(A) of T3 is performed after T1 which have modified the X data item which **satisfy the Cascadeless recoverable schedule.**

But, R(Y) of T2 (read from memory) is performing before the commit of T1 which is modifying the data item Y which does **also satisfy the Cascadeless recoverable schedule.**

In other words, no dirty read is happening in the given schedule.

Therefore, S4 is Cascadeless recoverable schedule.

If we change the commit order as shown in given schedule:

S4':	**T1**	**T2**	**T3**
	R(X)		
			R(Z)

		R(Y)
		W(Y)
		Commit	

	W(X)		
			W(Z)
	W(Y) Commit		
			R(X)
			Commit

The **Schedule S4', now is strict recoverable** as all the read and write operations on a data item is done after commit of the transaction which modifying the same data item.

For instance, W(Y) of T1 is done after commit of T2 and R(X) of T3 is done after T1.

Answer: 3(b) (ii) Consider the following schedule with two transactions T1 and T2. 2M

T1	T2

R(A)
W(A)	

	W(A) Commit
Commit	

In this schedule, if a failure occurs between the two commits, then we again cannot rollback T2's W(A) operation, and if we rollback T1's operation and run only T1 separately, then the final value that will be reflected in the database will be T1's W(A), whereas our original intension was to store T2's W(A) in the database. Hence, the update from T2 would be lost.

(Example 1M. Explanation 1M)

Q4. Consider a relation EMP (emp_id: integer, ename: string, jod_id: string, dept_id: integer, salary: integer, bonus: integer), where bonus is initially NULL for all the employee records of EMP.

(a) Write a stored procedure which will update the bonus (by 10% of the salary) of those employees in the

Q.4 (a) Create or replace procedure p1(b out number, tot out number) as — 1M

```
cursor c1 is select emp_id, ename, dept_id,
salary, bonus from emp e1 where salary <
(select avg(salary) from emp where
dept_id = e1.dept_id group by dept_id) and
job_id not in ('manager', 'supervisor'); 2M
s number:=0;
~~b number:=0~~; ~~c~~ c number:=0;
begin
    for rec in c1 loop              1M
      b:= 0.10 *rec.salary;
     update emp set bonus=b where   1M
              emp_id = rec.emp_id;
      s:=s+b;      1M
      c:=c+1;      1M
   endloop;
     b:=s;
     tot:=c;
   end p1;
```

Testing (calling the procedure)

```
Declare
    a number;
    total number;
  begin
      p1(a, total) —— 1M
   dbms_output.put_line('total bonus='
                              ||a);

  dbms_output.put_line('total emp got
                       bonus='||total);
 end;
```

EMP table who earn less salaries than the average salary of their respective departments, and who are not working as ‘Manager’ or ‘Supervisor’ in their department. Finally, the procedure computes the total updated bonus and the number of the employees who received the bonus, and send it to the calling environment for printing them as output. **8M**

(b) Write a database trigger on the relation EMP to ensure that the job_id of any new employee being hired to department 40 or updated in department 40 is either ‘Salesperson’ or ‘Clerk’. Add exception handling and provide an appropriate message so that the update fails if the new job_id is not ‘Salesperson’ or ‘Clerk’. **2M**

Answer: a)

b) create or replace trigger tg_emp 2M

```
before insert or update of job_id on emp

for each row

begin

if inserting then

if :new.job_id not in (‘salesperson’, ‘cleark’) and :new.dept_id=40 then

raise_applicaion_error(-20001, ‘not possible’);

end if;

end if;

if updating then

if :new.job_id not in (‘salesperson’, ‘cleark’) and :old.dept_id=40 then

raise_applicaion_error(-20001, ‘not possible’);

end if;

end if;

end;
```

Q5. Given the following four relational schemas, and answer the following SQL queries. (*Note*: *No partial marking will be considered for any of the SQL queries.*) **(5 x 2 = 10M)**

Employee (<u>ssn</u>: integer, fname: string, lname: string, bdate: date, Address: string, salary: integer, supervisor_ssn: integer, dno: integer), where supervisor_ssn is the self-referential foreign key and dno is the foreign key referencing to Department.

Department (<u>dno</u>: integer, dname: string, Mgr_ssn: integer, Mgr_start: date), where dname has unique constraint and Mgr_ssn is the foreign key referencing to Employee.

Project (<u>pno</u>: integer, pname: string, plocation: string, dno: integer), where pname has unique constraint and dno is the foreign key referencing to Department.

Works_on (<u>essn:</u>integer, <u>pno</u>: integer, hours: time), where essn is the foreign key referencing to Employee and pno is the foreign key referencing to Project.

(i)Find the average salary of those employees who work on the projects managed by their own department's manager.

Ans:
```
SELECT AVG(E.Salary) AS AvgSalary

FROM EMPLOYEE E

WHERE E.Dno =(

   SELECT D.Dno

   FROM DEPARTMENT D

   WHERE D.Mgr_ssn = E.Ssn

)

AND E.Ssn IN (

   SELECT W.Essn

   FROM WORKS_ON W

   JOIN PROJECT P ON W.pno = P.pno

   JOIN DEPARTMENT D ON P.dno = D.dno

   WHERE D.Mgr_ssn = E.Ssn

               );
```

(ii)Find the total number of hours worked by employees on projects managed by supervisors with a salary higher than $100,000.

Ans:
```
SELECT SUM(W.Hours) AS TotalHours

FROM WORKS_ON W

WHERE W.Pno IN (

   SELECT P.pno

   FROM PROJECT P

   JOIN DEPARTMENT D ON P.dno = D.dno

   JOIN EMPLOYEE M ON D.Mgr_ssn = M.Ssn

   WHERE M.Salary > 100000

);
```

(iii) Find the departments where the average employee salary is greater than the average salary of departments managed by supervisors with a salary greater than $80,000.

Ans:
```
SELECT D1.Dname

FROM DEPARTMENT D1

WHERE (

   SELECT AVG(E1.Salary)

   FROM EMPLOYEE E1

   WHERE E1.dno = D1.dno) >

(SELECT AVG(E2.Salary)

   FROM DEPARTMENT D2

   JOIN EMPLOYEE E2 ON D2.Mgr_ssn = E2.Ssn

   WHERE E2.Salary > 80000);
```

(iv)List the names of projects with at least one employee who works more than 40 hours on that project.

Ans:
```
SELECT DISTINCT P.Pname

FROM PROJECT P

WHERE EXISTS (

   SELECT *

   FROM WORKS_ON W

   WHERE W.Pno = P.Pno AND W.Hours > 40);
```

(v)List the names of employees who work on more than one project.

Ans:
```
SELECT DISTINCT E.Fname, E.Lname

FROM EMPLOYEE E

WHERE EXISTS (

   SELECT *

   FROM WORKS_ON W

   WHERE W.Essn = E.Ssn

   GROUP BY W.Essn

   HAVING COUNT(*) > 1

);
```

==