Assignment-2

1) Consider the following relational schemas....

a) Employee (ename, ecity, state)

b) Works (ename, company name, salary)

c) Company (company name, city)

d) Manager (ename, managername)

Write the expressions to solve the following queries using Relational Algebra, Tuple Relational Calculus and Domain relational Calculus.

a) Write the expression to find the name of the employees getting salary more than 60,000 & working in TCS and WIPRO.

Ans → Relational Algebra → Π[ename] (σ[salary > 60000 $\wedge$ (company_name = 'TCS' $\vee$ company_name = 'WIPRO')] (Works))

Tuple Relational Calculus → {w.name | Works(w) $\wedge$ w.salary > 60000 $\wedge$ (w.company_name = 'TCS' $\vee$ w.company_name = 'WIPRO')}

Domain Relational Calculus → {<ename> | $\exists$ company_name, salary (Works (ename, company_name, salary) $\wedge$ salary > 60000 $\wedge$ (company_name = 'TCS' $\vee$ company_name = 'WIPRO'))}

b) Write the expressions to find the salary and name of the employees working in any company placed in DELHI.

Ans → Relational Algebra → Π[ename, salary] (σ[city = 'DELHI'] (Company) $\bowtie$ Company.company_name = Works.company_name Works)

Tuple Relational Calculus (TRC) → {<w.ename, w.salary> | Work(w) $\wedge$ Company(c) $\wedge$ w.company_name = c.company_name $\wedge$ c.city = 'DELHI'}

Domain Relational Calculus (DRC) → {<ename, salary> | $\exists$ company_name, city (Works (ename, company_name, salary) $\wedge$ Company (Company_name, city) $\wedge$ city = 'DELHI')}

c) Write the expression to find the salary of the employees living in city MUMBAI and working under the manager JOHN.

Relational Algebra →

Ans → Π [salary] (σ [ecity = 'MUMBAI'] (Employee) $\bowtie$ Employee.ename = Works.ename Works $\bowtie$ Works.ename = Manager.ename σ [managername = 'JOHN'] (Manager))

Tuple Relational Calculus → $\{ \langle w.salary \rangle \mid Works(w) \wedge Employee(e) \wedge Manages(m) \wedge w.ename = e.ename \wedge w.ename \wedge e.ecity = 'MUMBAI' \wedge m.managername = 'JOHN' \}$?

Domain Relational Calculus → $\{ \langle salary \rangle \mid \exists ename, ecity, state, company\text{-}name, managername ($
Employee(ename, ecity, state) $\wedge$ ecity = 'MUMBAI' $\wedge$
Works(ename, company-name, salary) $\wedge$
Manages(ename, managername) $\wedge$ managername = 'JOHN') $\}$?

2) Consider the following three relations in a relational database. Employee(eId, Name), Brand(bId, bName), Own(eId, bId). Write the relational algebra expression that will return the set of eIds who own all the brands. Write the equivalent tuple relational calculus & domain relational calculus query.

Ans → Relational Algebra → Let take $R = \pi$ eId, bId , $S = \pi$ bId

π[eId](Own) − π[eId]((π[eId, bId](Own) × π[bId](Brand)) − Own)

Tuple Relational Calculus → $\{ e.eId \mid Employee(e) \wedge \forall b (Brand(b) \rightarrow \exists o (Own(o) \wedge o.eId = e.eId \wedge o.bId = b.Id)) \}$?

Domain Relational Calculus → $\{ \langle eId \rangle \mid \exists name (Employee(eId, name)) \wedge \forall bId, bName (Brand(bId, bName) \rightarrow \exists eId2 (Own(eId2, bId) \wedge eId2 = eId)) \}$?

3) Check whether the given schedules are conflict serializable or not-

a) S1: r1(x) r1(y) r2(x) r2(y) w2(y) w1(x)

b) S2: r1(x) r2(x) r2(y) w2(y) r1(y) w1(x)

Let ri(x) and wi(x) denote read and write operations respectively on a data item x by a transaction Ti. Consider the following two schedules.

Ans → a) Schedule S1 → r1(x) r1(y) r2(x) r2(y) w2(y) w1(x) Transactions → T1 & T2

Conflicts →

i) r1(x) & w1(x) → same T1 → no conflict

ii) r2(x) & r1(x) → read-read → no conflict

iii) r2(x) & w1(x) → r2(x) before w1(x) → conflict → T2 → T1

iv) r1(y) & w2(y) → T1 reads before T2 writes → conflict → T1 → T2

v) r2(y) & w2(y) → same T2 → no conflict

Precedence Graph Edges → T1 → T2 (because of r1(y) before w2(y))
T2 → T1 (because of r2(x) before w1(x))

Cycle detected → T1 → T2 → T1

∴ S1 is not conflict serializable.

b) Schedule S2 → S2: r1(x) r2(x) r2(y) w2(y) r1(y) w1(x)

Conflict → i) r1(x) & r2(x) → read-read → no conflict
ii) r2(y) & r1(y) → both read → no conflict
iii) w2(y) & r1(y) → w2(y) before r1(y) → conflict → T2 → T1
iv) r1(x) & w1(x) → same T1 → no conflict
v) r2(x) & w1(x) → T2 reads before T1 writes → conflict → T2 → T1

Precedence Graph Edges → T2 → T1 (from both w2(y) before r1(y) and r2(x) before w1(x))

No edge from T1 to T2 → no cycle

∴ S2 is conflict serializable

4) Check whether the given schedule below is conflict serializable or not justify your answer & show the precedence graph.

T_1	T_5
read(A)	
A := A-50	
write(A)	
	read(B)
	B := B-10
	write(B)
read(B)	
B := B+50	
write(B)	
	read(A)
	A := A+10
	write(A)

Ans → Conflict Detection →

i) On B → a) T5: write(B) → T1: read(B) = W-R conflict → T5 → T1

b) T5: write(B) → T1: write(B) = W-W conflict → T5 → T1

Name: _____ Regd No: _____

~~Conflicts~~ ii) On A → a) T1 : write(A) → T5 : read(A) = W-R conflict → T1 → T5

b) T1 : write(A) → T5 : write(A) = W-W conflict → T1 → T5

Precedence Graph → Nodes → T1, T5

Edges → T1 → T5 (due to conflict on A)

T5 → T1 (due to conflict on B)

Cycle detected → T1 → T5 → T1

Conclusion → The schedule is not conflict serializable because the precedence graph contains a cycle.

5) Check whether the given schedule S is conflict serializable and recoverable or not.

T1	T2	T3	T4
	READ(x)		
		WRITE(x)	
		COMMIT	
WRITE(x)			
COMMIT			
	WRITE(y)		
	READ(z)		
	COMMIT		
			READY(x)
			READY(y)
			COMMIT

Schedule S

Ans→ Conflicts on X →

i) T2 : READ(x) before T3 : WRITE(x) → T2 → T3

ii) T3 : WRITE(x) before T1 : WRITE(x) → T3 → T1

iii) T1 : WRITE(x) before T4 : READ(x) → T1 → T4

Conflicts on Y → T2 : WRITE(Y) before T4 : READ(y) → T2 → T4

Build Precedence Graph → Nodes → T1, T2, T3, T4

Edges → T2 → T3 T2 → T4

T3 → T1

T1 → T4 The graph has no cycles.

Conflict Serializable → Because the precedence graph is acyclic, the schedule is conflict serializable

Possible serial order → T2 → T3 → T1 → T4

Recoverability → A schedule is recoverable if a transaction only commits after the transactions it reads from have committed.

- T2 reads X before T3 writes it → No read-after-write conflict
- T4 reads X written by T1, and Y written by T2 → T4 commits after both T1 & T2

Yes, its recoverable

6) Consider a schedule of Transaction T1 and T2:

```
T1  RA        RC        WD        WB  Commit
T2      RB  WB      RD      WC          Commit
```

Here, RX stands for "Reads(X)" and WX stands for "Write(x)". Which one of the following schedules is conflict equivalent to the above schedule?

a) T1 RA RC WD WB Commit
   T2 RB WB RD WC Commit

b) T1 RA RC WD WB Commit
   T2 RB WB RD WC Commit

Ans → From the given schedule, Conflicts → • Read-Write (RW) on the same item
• Write-Read (WR) on same item
• Write-Write (WW) on same item.

1) T1: WD and T2: RD → WR conflict on D → T1 → T2
2) T1: WB and T2: WB → WW conflict on B → T1 → T2
3) T2: WB and T1: WB → WW conflict on B → T2 → T1

So, this gives a cycle → T1 → T2 → T1 → Not conflict serializable

∴ The original schedule is not conflict serializable.

In option →
a) Same as the original schedule ⇒ Also not conflict serializable.

b) Conflicts → i) T2: WB before T1: WB → WW conflict → T2 → T1
ii) T1: WD before T2: RD → WR conflict → T1 → T2

Name: _____ Regd No: _____

→ T1 → T2 and T2 → T1 ⇒ Cycle exists, not serializable

So,
Neither (A) nor (B) is conflict equivalent to a conflict-serializable schedule.

7) Consider the following two transactions →

```
T1 : read(A);                    T2 : read(B);
     read(B);                         read(A);
     if A=0 then B:=B+1;              if B=0 then A:=A+1;
     write(B).                        write(A).
```

Add lock and unlock instructions to transactions T1 & T2, so that they observe the two phase locking protocol. Can the execution of these transactions result in a deadlock?

Ans→ Apply Two-Phase Locking (2PL) → • Growing phase: Aquire all locks (shared or exclusive)- no unlocks.
• Shrinking phase: Release locks - no new locks allowed.

We assume that : • Shared Locks(S) for reading.
• Exclusive Lock (X) for writing.

T1 with locks →	T2 with Locks →
LOCK-S(A)	LOCK-S(B)
read(A)	LOCK-S(A)
LOCK-S(B)	LOCK-X(A)
read(B)	read(B)
UNLOCK-S(A)	read(A)
if A==0 then	if B==0 then
LOCK-X(B)	A:=A+1
B:=B+1	write(A)
write(B)	UNLOCK-X(A)
UNLOCK-X(B)	UNLOCK-X(B)
UNLOCK-S(B)	UNLOCK-X(C)

Yes - a classic deadlock situation is possible.

Suppose :
• T1 locks A (shared), then requests lock on B
• T2 locks B (shared), then request lock on A

Then :
• T1 is waiting for B (held by T2)
• T2 is waiting for A (held by T1) ∴ This is a circular wait ⇒ Deadlock

8) Check whether the given schedule below can complete all its operations in the given order using Timestamp based protocol or not.

T1	T2
READ(A)	
	READ(A)
WRITE(A)	
	WRITE(A)

Ans → Time Stamp Protocol rules →

For a data item, X:

Read rule:

- If TS(T) < W_TS(X) → Abort T (younger T is trying to read outdated value)
- Else → allow, and set R_TS(X) = max(R_TS(X), TS(T))

Write rule:

- If TS(T) < R_TS(X) → Abort T
- If TS(T) < W_TS(X) → Abort T
- Else → allow, and update W_TS(X)

Lets denote: Initial: R_TS(A)=0, W_TS(A)=0

Step 1: T1 reads A

- Allowed since TS(T1) < W_TS(A) is false (T1 is older)
- Update R_TS(A) = TS(T1)

Step 2: T2 reads A

- Allowed if TS(T2) >= W_TS(A) → true (W_TS(A)=0)
- Update R_TS(A) = TS(T1);

Step 3: T1 writes A

- Allowed if: TS(T1) >= R_TS(A) may be false now (TS(T1) < TS(T2) == R_TS(A)) → Abort T1

So T1 will fail to write A.

Step 4: T2 writes A

- T2 is newer, R_TS(A) = TS(T2), so
- TS(T2) >= R_TS(A)
- TS(T2) >= W_TS(A)
- → Allowed

∴ No, the schedule cannot complete all operations using timestamp-based protocol.

9) Consider T_0 & T_1 be a two transaction. To transfer 50rs from account A to account B and T_1 withdraw 100rs from account C. Both transaction are commit successfully. Write down the system log corresponding to T_0 & T_1 to handle any failures.

Name: _____ Regd No: _____

Ans→ Initial bal → A=1000, B=2000, C=700

Transactions → T0: Transfer ₹50 from A to B ; T1: Withdraw ₹100 from C

→ A = A - 50 → 950

→ B = B + 50 → 2050

→ C = C - 100 → 600

System Log →
- START T0
- <T0, X, old_value, new_value>
- Commit T0

Log Entries → START T0

<T0, A, 1000, 950>

<T0, B, 2000, 2050>

COMMIT T0

START T1

<T1, C, 700, 600>

COMMIT T1

10) Write down Two major differences between immediate and deffered database modification.

Ans→ i) When the changes are applied to the database:

- Immediate Modification → Changes to the database are applied as soon as the write operation is issued, even before the transaction commits.
- Deffered Modification → Changes are not applied immediately; they are deffered until the transaction successfully commits.

ii) Logging Requirements →

- Immediate Modification → Requires both undo & redo information in the log (since changes happen before commit and need might need to be undone if a crash occure).
- Deffered Modification → Requires only redo information (since changes are applied only after a successful commit, no undo is neccesary).

Name: _____ Regd No: _____