National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Advance Database Concepts**	**Course Code:**	**CS 4064**
Program:	**BS (SE)**	**Semester:**	**Spring 2025**
Due Date:	**18th of February, 2025.**	**Total Marks:**	**100**
Section:	**6A, 8A**	**Page(s):**	**4**
Type:	**Assignment 1**		

Important Instructions:

The assignment MUST be hand written and submitted in hard form. Use your OWN roll numbers where required. Show all steps/tables and variable states and locks in each part.

You are given the following **base transaction schedule**:

S: r2(X), w1(Y), w2(Y), w1(X), w3(Z), c1, w2(X), c2, w3(X), c3

(Transactions: **T1, T2, T3**)

Modify this base schedule according to the last **four digits** of your **roll number (XXXX)** using the **modification table** provided below. Each digit (0-9) corresponds to a specific modification rule, and you must apply **four modifications in sequence**, one for each digit in your roll number.

1. **Start with the given base schedule.**
2. **Apply the first modification** based on the first digit of your roll number.
3. **Use the modified schedule as the new base schedule** for the next modification.
4. Continue this process until you have applied **all four modifications**.

If a modification requires adding a new transaction (T4), but T4 has already been introduced in an earlier step, use the same T4 instead of creating a new one.

After completing the modifications, answer the provided questions based on your **final** modified schedule.

Modification Table Based on Roll Number Digits

Digit (0-9)	Specific Modification Instruction
0	**Introduce a new transaction T4** with r4(X). Insert r4(X) **immediately after the first read operation (r2(X))** in the schedule.

1	**Introduce a new transaction T4** with w4(Y). Insert w4(Y) **immediately before the first commit (c1)** in the schedule.
2	**Swap actions of T2 and T3**: Swap the first action of T2 (r2(X)) with the first action of T3 (w3(Z)).
3	**Introduce a cascading rollback**: Insert abort(T3) **immediately after w3(Z), and remove all of T3's subsequent operations (c3, w3(X)).**
4	**Force a conflict by changing the isolation level**: Change r2(X) to w2(X), creating a **write-write conflict** with w1(X).
5	**Introduce a phantom read**: Insert r3(W) (T3 reads a new item W) **immediately before w3(Z).**
6	**Introduce a deadlock scenario**: Insert lock(Y) by T1 before w1(Y), and lock(Y) by T2 before w2(Y), forcing a deadlock.
7	**Force a timestamp violation**: Change w2(X) to occur **before** w1(X), violating timestamp ordering rules.
8	**Introduce a new transaction T4** with r4(Y), w4(Y), c4. Insert T4 **right after w2(Y), but before w1(X).**
9	**Force a non-serializable schedule**: Swap w3(X) and w2(X), leading to an **incorrect final state** (non-serializable execution).

1. **Write down your modified schedule** and solve the following **for your custom schedule**:

(a) Analyze how each of the following concurrency control techniques processes your custom schedule. Provide step-by-step actions, blocking cases, deadlocks, and resolutions where applicable.

(b) **Show the execution trace including lock requests, timestamps, waits, and rollbacks.**

(c) If a transaction is aborted, explain how the system handles it under each technique.

- Basic 2PL with **Wait-Die**
- Strict 2PL with **Wound-Wait**
- Rigorous 2PL with **Wait-Die**
- Rigorous 2PL with **Wound-Wait**
- Rigorous 2PL with **Deadlock Detection (Wait-for-Graph)**
- **Basic Timestamp Ordering**
- **Strict Timestamp Ordering**
- **Timestamp Ordering with Thomas's Write Rule (TWR)**
- **Multi-Version Timestamp Ordering**
- **Validation (Optimistic Concurrency Control)**

Part 2: Multi-Version Concurrency Control (MVCC) Analysis
Using your customized schedule
Determine how MVCC would handle your transactions.

- How many versions of each variable (X, Y, Z) will exist?
- When does a transaction read from an older version?
- Show a timeline of how the versions evolve.

2. **Modify your schedule** by inserting a **phantom read scenario** and analyze how **MVCC handles it differently from 2PL.**
(E.g., Add a new transaction T4/T5 that inserts a new record and see how snapshot isolation prevents anomalies.)

3. **Part 3: Recovery Mechanisms & Crash Handling**

1. Suppose a system crash occurs **immediately after w2(X) in your custom schedule**. Analyze the following:

   - **Which transactions need to be undone?**
   - **Which transactions can be redone?**
   - **How would ARIES (Advanced Recovery Algorithm) handle this?**
   - **Compare it with a simple undo-redo recovery mechanism.**

2. **Modify your schedule** so that **T1 is a long-running transaction** and crashes before its commit.
    - How does **Strict 2PL** behave differently from **Basic Timestamp Ordering** in this case?
    - Which **recovery strategy** is more efficient here?

Part 4: Serializability and Isolation Levels
For your custom schedule:
Determine whether the schedule is conflict-serializable.

- Construct a **precedence graph** and check for cycles.

2. **Analyze the impact of different isolation levels:**
    - Read Uncommitted
    - Read Committed
    - Repeatable Read
    - Serializable

For each level, identify which anomalies may or may not occur.

----------------------*Best of Luck*-------------------