1. (20 points) Consider a database that has two integer objects, A and B. DBMS is given two programs:

| P1 | P2 |
|---|---|
| A = A + 1<br>B = B + 1 | A = B<br>B = B + 1 |

Before the program is execution, A = 0 and B = 0. For each of the following schedules, determine if it is a serializable schedule. You must explain why.

S1

| T1 | T2 |
|---|---|
|  | A = B |
| A = A + 1 |  |
| B = B + 1 |  |
|  | B = B + 1 |

1. **S1**: Yes. A schedule is a serializable schedule if its effect on the database is identical to that of some serial schedule.
   **S2**: No. A = 0 after operating this schedule
   **S3**: Yes. A schedule is a serializable schedule if its effect on the database is identical to that of some serial schedule.

S2

| T1 | T2 |
|---|---|
| A = A + 1 |  |
|  | A = B |
| B = B + 1 |  |
|  | B = B + 1 |

S3

| T1 | T2 |
|---|---|
|  | A = B |
| A = A + 1 |  |
|  | B = B + 1 |
| B = B + 1 |  |

2. (20 points) Consider the following two programs.

| P1 | P2 |
|---|---|
| R(A) | W(A) |
| R(C) | R(B) |
| W(C) | W(B) |
| c/o | c/o |

For each of the following execution, determine if it is a schedule, a serial schedule, and/or a strict schedule. You must explain why.

S1

| T1 | T2 |
|---|---|
| R(A) | |
| | W(A) |
| R(C) | |
| | R(B) |
| W(C) | |
| | W(B) |
| c/o | |
| | c/o |

S2

| T1 | T2 |
|---|---|
| R(A) | |
| R(C) | |
| W(C) | |
| | W(A) |
| c/o | |
| | R(B) |
| | W(B) |
| | c/o |

S3

| T1 | T2 |
|---|---|
| R(A) | |
| R(C) | |
| W(C) | |
| c/o | |
| | W(A) |
| | R(B) |
| | W(B) |
| | c/o |

S4

| T1 | T2 |
|---|---|
| | W(A) |
| R(A) | |
| W(C) | |
| c/o | |
| | R(B) |
| | W(B) |
| | c/o |

2. **S1:** schedule. T2 W(A) is before T1 R(A) c/o
   **S2:** schedule. r/w (A) conflict
   **S3:** serial schedule. There is no interleave.
   **S4:** schedule. r/w (A) conflict

3. (40 points) Consider the following two schedules implemented by strict 2PL.

S1

| T1 | T2 |
|---|---|
| S(A) | |
| R(A) | |
| X(A) | |
| W(A) | |
| commit | |
| | S(A) |
| | R(A) |
| | commit |

S2

| T1 | T2 |
|---|---|
| S(A) | |
| R(A) | |
| | S(A) |
| | R(A) |
| X(A) | |
| | commit |
| W(A) | |
| commit | |

Recall strict 2PL is implemented with a lock table to keep 1) which data object is locked; 2) which transaction owns which lock; 3) which transaction is waiting for a lock on this object.

Lock table

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| | | | |

For each of the two schedules, explain how the status of the lock table changes as the actions are executed, by filling the tables in the next two pages.

S1

| T1 | T2 |
|---|---|
| S(A)<br>R(A)<br>X(A)<br>W(A)<br>commit | |
| | S(A)<br>R(A)<br>commit |

1) T1.S(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | S | T1 | T2 |

2) T1.R(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | S | T1 | T2 |

3) T1.X(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | X | T1 | T2 |

4) T1.W(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | X | T1 | T2 |

5) T1.commit

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | | | |

6) T2.S(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | S | T2 | |

7) T2.R(A)

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | S | T2 | |

8) T2.commit.

| Data | Lock | Owner | Waiting |
|---|---|---|---|
| A | | | |

S2

| T1 | T2 |
|---|---|
| S(A) | |
| R(A) | |
| | S(A) |
| | R(A) |
| X(A) | |
| | commit |
| W(A) | |
| commit | |

1) T1.S(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | S | T1 | T2 |

2) T1.R(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | S | T1 | T2 |

3) T2.S(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | S | T2 | |

4) T2.R(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | S | T2 | |

5) T1.X(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | X | T1 | T1 |

6) T2.commit.

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | | | |

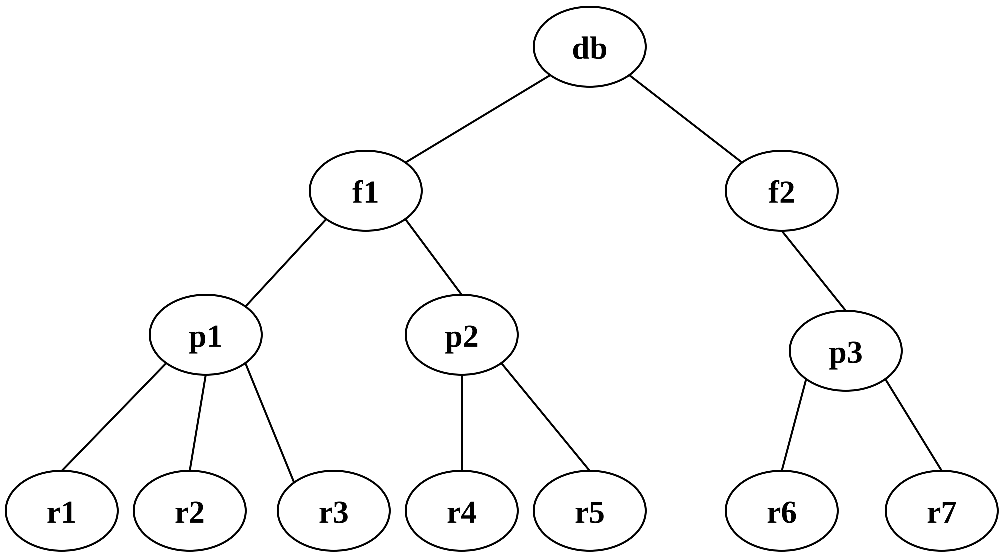
7) T1.W(A)

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | X | T1 | |

8) T1.commit

| *Data* | *Lock* | *Owner* | *Waiting* |
|---|---|---|---|
| A | | | |

4. (20 points) For better current execution, multiple-granularity locking (MGL) introduces three new types of locks, IS, IX, and SIX. Consider the following tree of objects, where each node contains all its children.



a) (6 points) Transaction T1 needs to read object r1. What lock(s) on which node(s) does it need to acquire?

Db&F1&P1 -> IS

r1 -> S

a) (6 points) Transaction T2 needs to write object r4 and r5. What lock(s) on which node(s) does it need to acquire?

Db&F1&P2 -> IX

r4&r5 -> x

a) (8 points) Transaction T3 needs to read r6 and r7. What lock(s) should it acquire on object p3?

IS