Name: Mareena Fernandes
Roll no.: 8669
Class: TE IT
Batch: B

Experiment No.: 4

timestamp_log.json:


```
{
  "log": [
    {
      "transaction": 25,
      "timestamp": 25,
      "operation": "read",
      "file": "B"
    },
    {
      "transaction": 26,
      "timestamp": 26,
      "operation": "read",
      "file": "B"
    },
    {
      "transaction": 26,
      "timestamp": 26,
      "operation": "write",
      "file": "B"
    },
    {
      "transaction": 25,
      "timestamp": 25,
      "operation": "read",
      "file": "A"
    },
    {
      "transaction": 26,
      "timestamp": 26,
      "operation": "read",
      "file": "A"
    },
```

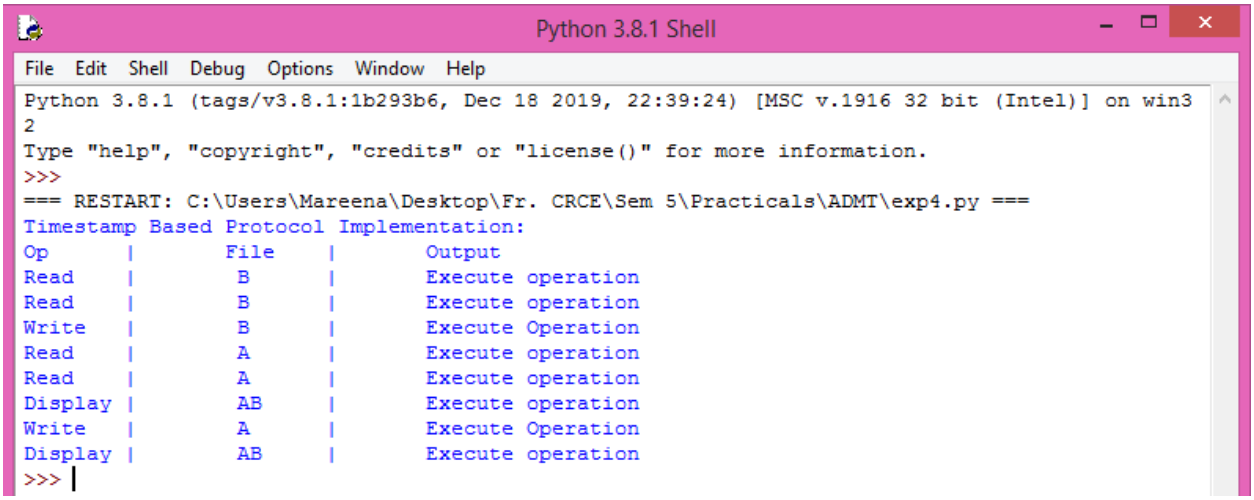
```
    {
      "transaction": 25,
      "timestamp": 25,
      "operation": "display",
      "file": "AB"
    },
    {
      "transaction": 26,
      "timestamp": 26,
      "operation": "write",
      "file": "A"
    },
    {
      "transaction": 26,
      "timestamp": 26,
      "operation": "display",
      "file": "AB"
    }
  ]
}
```

exp4.py:

```
import json
from collections import defaultdict

file_map = {}
current_file = 0
files = []
transaction_map = {}
transactions = []

with open('./timestamp_log.json') as f:
    log = json.load(f)["log"]

    rts = [0] * len(log)
    wts = [0] * len(log)

    print("Timestamp Based Protocol Implementation: ")
    print("Op\t|\tFile\t|\tOutput")
    for data in log:
        if not data["file"] in files:
            file_map[data["file"]] = current_file
            current_file += 1
            files.append(data["file"])

        if not data["transaction"] in transactions:
            transaction_map[data["transaction"]] = data["timestamp"]
            transactions.append(data["transaction"])

        mapped_file_num = file_map[data["file"]]

        if data["operation"] == "read":
            if wts[mapped_file_num] > data["timestamp"]:
                transaction_num = list(transaction_map.keys())[list(
                    transaction_map.values()).index(wts[mapped_file_num])]

                print(
                    "Read\t|\t", data["file"], "\t|\tRollback and execute after Transaction", transaction_
num)

            elif data["timestamp"] > rts[mapped_file_num]:
                rts[mapped_file_num] = data["timestamp"]

                print("Read\t|\t", data["file"], "\t|\tExecute operation")
```

```
    elif data["operation"] == "write":
        if rts[mapped_file_num] > data["timestamp"]:
            transaction_num = list(transaction_map.keys())[list(
                transaction_map.values()).index(wts[mapped_file_num])]

            print(
                "Write\t|\t", data["file"], "\t|\tRollback and execute after Transaction", transaction
_num)

        elif wts[mapped_file_num] > data["timestamp"]:
            print(
                "Write\t|\t", data["file"], "\t|\tReject and rollback (Obsolete write)")

        else:
            wts[mapped_file_num] = data["timestamp"]

            print(
                "Write\t|\t", data["file"], "\t|\tExecute Operation")

    elif data["operation"] == "display":
        print("Display\t|\t", data["file"], "\t|\tExecute operation")

    else:
        print("Invalid operation type")
```

Output:



Post labs:

1. Consider the transactions T1, T2, and T3 and the schedules S1 and S2 given below

T1: r1(X); r1(Z); w1(X); w1(Z)
T2: r2(Y); r2(Z); w2(Z)
T3: r3(Y); r3(X); w3(Y)
S1: r1(X); r3(Y); r3(X); r2(Y); r2(Z);
w3(Y); w2(Z); r1(Z); w1(X); w1(Z)
S2: r1(X); r3(Y); r2(Y); r3(X); r1(Z);
r2(Z); w3(Y); w1(X); w2(Z); w1(Z)

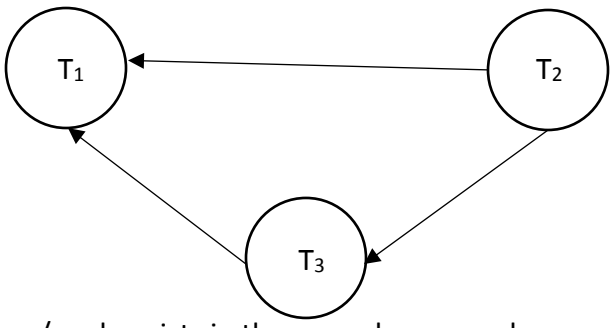
Which one of the schedules S1, S2 is conflict serializable?

Ans:

Given Schedule S_1:

T_1	T_2	T_3
$r_1(x)$		
		$r_3(y)$
		$r_3(x)$
	$r_2(y)$	
	$r_2(z)$	
		$w_3(y)$
	$w_2(z)$	
$r_1(z)$		
$w_1(x)$		
$w_1(z)$		

Precedence Graph:



Since no loop/ cycle exists in the precedence graph.

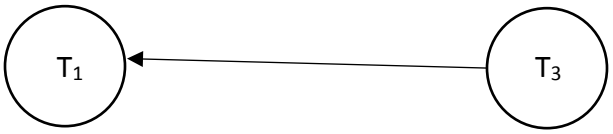
∴ Given schedule S_1 is conflict serializable.

To make Serial Schedule:

Indegree $T_1 = 2$
Indegree $T_2 = 0$
Indegree $T_3 = 1$

$\therefore$ Remove T_2 from graph and add it in sequence.

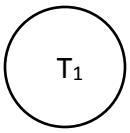
Precedence Graph:



Indegree $T_1 = 1$
Indegree $T_3 = 0$

$\therefore$ Remove T_3 from graph and add it in sequence.

Precedence Graph:



Indegree $T_1 = 0$

$\therefore$ Sequence for serial schedule is,

$[T_2 \rightarrow T_3 \rightarrow T_1]$

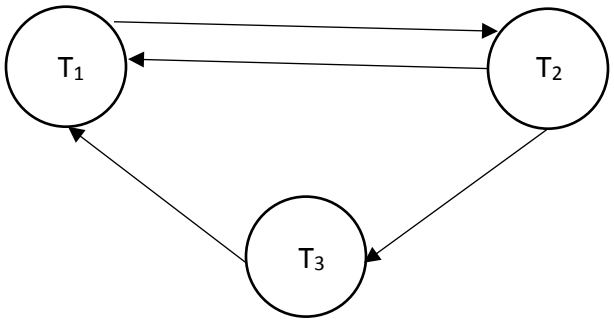
T_1	**T_2**	**T_3**
	$r_2(y)$	
	$r_2(z)$	
	$w_2(z)$	
		$r_3(y)$
		$r_3(x)$
		$w_3(y)$
$r_1(x)$		
$r_1(z)$		
$w_1(x)$		
$w_1(z)$		

Serial Schedule S_1'

Given Schedule S_2:

T_1	T_2	T_3
$r_1(x)$		
		$r_3(y)$
	$r_2(y)$	
		$r_3(x)$
$r_1(z)$		
	$r_2(z)$	
		$w_3(y)$
$w_1(x)$		
	$w_2(z)$	
$w_1(z)$		

Precedence Graph:



Since there exists a cycle in the precedence graph.

$\therefore$ Given schedule S_2 is not conflict serializable.

2. List the disadvantages of time stamp protocol. And explain what is Thomas write rule.

Ans:

- Recoverability:
  In timestamp-based protocol, since the only limiting factor is the timestamp being the correct value, it is possible that a transaction reads data from an uncommitted write which a failure would be unrecoverable. This causes cascading rollback to that pointing time thus wasting time and computer power timestamp resolution. If minimum time elapsed between multiple transactions is less than resolution or coarseness of timestamp, it is possible to assign some timestamp to multiple transactions.
- Thomas Write Rule:
  In timestamp-based protocol, if any operations don't follow order of serializability, it is rejected and rollback. Some operations can be harmless though and do not require rollback. Thomas Write Rule identifies updated writes to be harmless and thus instead of rolling back, simply ignores the operation moving on the next.
  a) If RTS(X) > TS(Ti), then abort and rollback to ensure consistency.
  b) If WTS(X) > TS(Ti), don't execute write operation and continue since even if performed before timestamp, it would be overwritten in future thus making it outdated or obsolete write.

Experiment No.: 5

aries.json:

```
{
 "log": [
   {
     "lsn": 1,
     "operation": "write",
     "transaction": 1,
     "file": "A",
     "page": 1
   },
   {
     "lsn": 2,
     "operation": "write",
     "transaction": 2,
     "file": "B",
     "page": 1
   },
   {
     "lsn": 3,
     "operation": "write",
     "transaction": 2,
     "file": "C",
     "page": 2
   },
   {
     "lsn": 4,
     "operation": "flush",
     "page": 2
   },
   {
     "lsn": 5,
     "operation": "write",
     "transaction": 1,
     "file": "D",
     "page": 2
   },
   {
     "lsn": 6,
     "operation": "commit",
     "transaction": 1
```

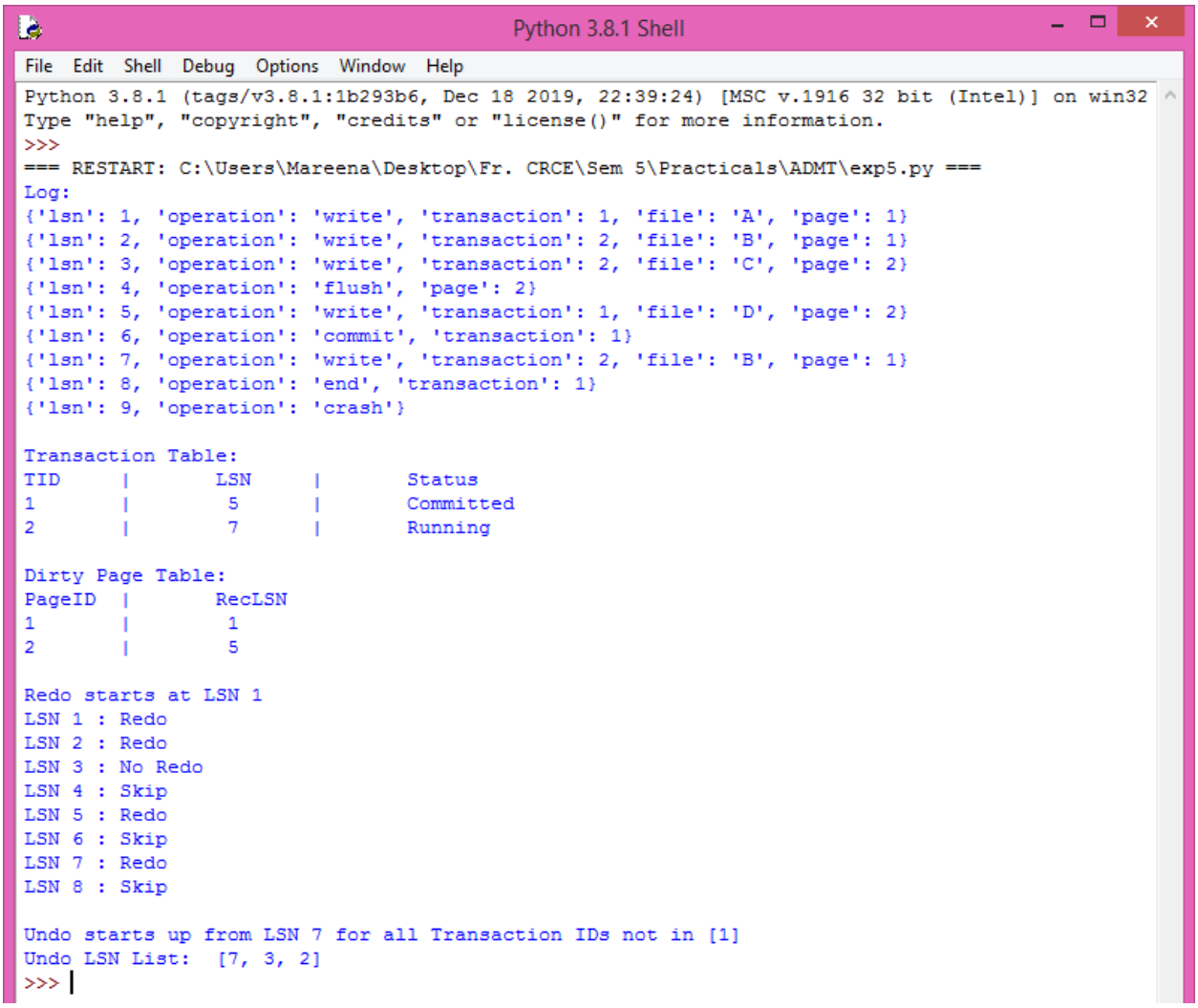
```
    },
    {
      "lsn": 7,
      "operation": "write",
      "transaction": 2,
      "file": "B",
      "page": 1
    },
    {
      "lsn": 8,
      "operation": "end",
      "transaction": 1
    },
    {
      "lsn": 9,
      "operation": "crash"
    }
  ]
}
```

exp5.py:

```
import json
from collections import defaultdict

dirty_rec = defaultdict(lambda: 0)
trans_rec = defaultdict(lambda: 0)
page_rec = defaultdict(lambda: 0)

committed = []

undo_lsn = []
first_redo = float("inf")
last_undo = 0

with open('./aries.json') as f:
    log = json.load(f)["log"]

    print("Log: ")
    for data in log:
        print(data)
        if data["operation"] == "read" or data["operation"] == "write":
            trans_rec[data["transaction"]] = data["lsn"]
            if dirty_rec[data["page"]] == 0:
                dirty_rec[data["page"]] = data["lsn"]

        elif data["operation"] == "commit":
            if not data["transaction"] in committed:
                committed.append(data["transaction"])

        elif data["operation"] == "flush":
            dirty_rec[data["page"]] = 0
            page_rec[data["page"]] = data["lsn"]

        elif data["operation"] == "crash":
            break

        elif data["operation"] == "end":
            continue

        else:
            print("Invalid log file. Illegal operation type")
            break
```

```
    print("\nTransaction Table: ")
    print("TID\t|\tLSN\t|\tStatus")
    for key, val in trans_rec.items():
        print(key, "\t|\t", val, end="\t|\t")
        if key in committed:
            print("Committed")
        else:
            if val > last_undo:
                last_undo = val
            print("Running")

    print("\nDirty Page Table: ")
    print("PageID\t|\tRecLSN")
    for key, val in dirty_rec.items():
        if val != 0:
            print(key, "\t|\t", val)
            if val < first_redo:
                first_redo = val

    print("\nRedo starts at LSN", first_redo)

    for i in range(first_redo - 1, len(log)):
        data = log[i]
        if data["operation"] == "read" or data["operation"] == "write":
            if dirty_rec[data["page"]] > 0 and dirty_rec[data["page"]] <= data["lsn"] and page_rec[da
ta["page"]] < data["lsn"]:
                print("LSN", data["lsn"], ": Redo")
            else:
                print("LSN", data["lsn"], ": No Redo")
        elif data["operation"] == "crash":
            continue
        else:
            print("LSN", data["lsn"], ": Skip")

    print("\nUndo starts up from LSN", last_undo,
          "for all Transaction IDs not in", committed)

    for i in range(last_undo - 1, -1, -1):
        data = log[i]
        if data["operation"] == "read" or data["operation"] == "write":
            if not data["transaction"] in committed:
                undo_lsn.append(data["lsn"])

    print("Undo LSN List: ", undo_lsn)
```

Output:


Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\Mareena\Desktop\Fr. CRCE\Sem 5\Practicals\ADMT\exp5.py ===
Log:
{'lsn': 1, 'operation': 'write', 'transaction': 1, 'file': 'A', 'page': 1}
{'lsn': 2, 'operation': 'write', 'transaction': 2, 'file': 'B', 'page': 1}
{'lsn': 3, 'operation': 'write', 'transaction': 2, 'file': 'C', 'page': 2}
{'lsn': 4, 'operation': 'flush', 'page': 2}
{'lsn': 5, 'operation': 'write', 'transaction': 1, 'file': 'D', 'page': 2}
{'lsn': 6, 'operation': 'commit', 'transaction': 1}
{'lsn': 7, 'operation': 'write', 'transaction': 2, 'file': 'B', 'page': 1}
{'lsn': 8, 'operation': 'end', 'transaction': 1}
{'lsn': 9, 'operation': 'crash'}
Transaction Table:
TID | LSN | Status
1 | 5 | Committed
2 | 7 | Running
Dirty Page Table:
PageID | RecLSN
1 | 1
2 | 5
Redo starts at LSN 1
LSN 1 : Redo
LSN 2 : Redo
LSN 3 : No Redo
LSN 4 : Skip
LSN 5 : Redo
LSN 6 : Skip
LSN 7 : Redo
LSN 8 : Skip
Undo starts up from LSN 7 for all Transaction IDs not in [1]
Undo LSN List: [7, 3, 2]
>>>

Post labs:

1. Implement ARIES algorithm for the above given sample log file.

Ans:

Analysis Phase:

LSN	T_id	Prev_LSN	Operation	Page_id
01	T_1	-	Write(A)	P_1
02	T_2	-	Write(B)	P_1
03	T_3	02	Write(C)	P_2
FLUSH P_2 TO DISK				
05	T_1	01	Write(D)	P_2
06	T_1	05	Commit	-
07	T_2	03	Write(B)	P_1
08	T_1	06	End	-
09	-	-	Crash	-

Page LSN = P_2 = 03

Transaction Table:

T_id	LSN	Status
T_1	~~1~~ ~~5~~ 6	Committed
T_2	~~2~~ ~~3~~ 7	Running

Dirty Page Table:

Page_id	Rec_LSN
P_1	01
~~P_2~~	~~03~~
P_2	05

← Removed

Redo Phase:

LSN 01 – Redo

P_1 in DPT
Rec LSN ≤ Current LSN
Page LSN < Current LSN

LSN 02 – Redo

P_1 in DPT
Rec LSN ≤ Current LSN
Page LSN < Current LSN

LSN 03 – Redo

P_2 in DPT
Rec LSN ≤ Current LSN

LSN 04 – Skip {Flush}

LSN 05 – Redo

P_2 in DPT
Rec LSN ≤ Current LSN
Page LSN < Current LSN

LSN 06 – Skip {Commit}

LSN 07 – Redo

P_1 in DPT
Rec LSN ≤ Current LSN
Page LSN < Current LSN

LSN 08 – Skip {End}

LSN 09 – Skip {Crash}

<u>Undo Phase:</u>

Transaction Table:

T_id	**LSN**	**Status**
T_2	7	Running

Start Undo from LSN 7

LSN	T_id	Prev_LSN	Operation	Page_id
01	T_1	-	Write(A)	P_1
02	T_2	-	Write(B)	P_1
03	T_2	02	Write(C)	P_2
04	-	-	Flush P_2	-
05	T_1	01	Write(D)	P_2
06	T_1	05	Commit	-
07	T_2	03	Write(B)	P_1
08	T_1	06	End	-
09	T_2		Undo LSN: 07	P_1
10	T_2		Undo LSN: 03	P_2
11	T_2		Undo LSN: 02	P_1

Experiment No.: 6

```
CREATE TABLE pay (Title VARCHAR(20) PRIMARY KEY, Salary NUMERIC(6));

INSERT INTO pay VALUES ('Elect. Engg.', 40000);
INSERT INTO pay VALUES ('Syst. Analy.',50000);
INSERT INTO pay VALUES ('Mech. Engg.',42000);
INSERT INTO pay VALUES ('Programmer',65000);

SELECT * FROM pay;

CREATE TABLE emp (Eno NUMERIC(5) PRIMARY KEY, Ename VARCHAR(20), Title VARCHAR(20)
REFERENCES pay);

INSERT INTO emp VALUES (101,'John','Elect. Engg.');
INSERT INTO emp VALUES (102,'Sam','Syst. Analy.');
INSERT INTO emp VALUES (103,'Robert','Mech. Engg.');
INSERT INTO emp VALUES (104,'Kim','Programmer');
INSERT INTO emp VALUES (105,'Robert','Elect. Engg.');
INSERT INTO emp VALUES (106,'Jack','Syst. Analy.');
INSERT INTO emp VALUES (107,'Som','Programmer');
INSERT INTO emp VALUES (108,'Smith','Elect. Engg.');
INSERT INTO emp VALUES (109,'Albert','Mech. Engg.');
INSERT INTO emp VALUES (110,'Bolt','Mech. Engg.');

SELECT * FROM emp;
```

1. **Horizontal Fragmentation-**

   CREATE TABLE pay1 AS SELECT * FROM pay WHERE Salary<=45000;

   CREATE TABLE pay2 AS SELECT * FROM pay WHERE Salary>45000;

   SELECT * FROM pay1;

   SELECT * FROM pay2;

   SELECT * FROM pay1 UNION ALL SELECT * FROM pay2;

2. **Vertical Fragmentation-**

   CREATE TABLE emp1 AS SELECT Eno, emp.Ename FROM emp;

   SELECT * FROM emp1;

   CREATE TABLE emp2 AS SELECT Eno, emp.Title FROM emp;

   SELECT * FROM emp2;

   SELECT emp1.Eno, emp1.Ename, emp2.Title FROM emp1, emp2 WHERE emp1.Eno=emp2.Eno;

   SELECT emp1.Eno, emp1.Ename, emp2.Title FROM emp1 FULL OUTER JOIN emp2 ON emp1.Eno=emp2.Eno;

Post labs:

1. Implement derived Hybrid Fragmentation.

Ans:

SELECT Eno, Ename FROM emp WHERE Eno>=105;

SELECT Eno, Title FROM emp WHERE Title='Elect. Engg.';

CREATE TABLE emp_frag1 AS SELECT emp.Eno, emp.Ename, emp.Title FROM emp, pay1 WHERE emp.Title=pay1.Title;

SELECT * FROM emp_frag1;

CREATE TABLE emp_frag2 AS SELECT emp.Eno, emp.Ename, emp.Title FROM emp, pay2 WHERE emp.Title=pay2.Title;

SELECT * FROM emp_frag2;

SELECT * FROM emp_frag1 UNION ALL SELECT * FROM emp_frag2;