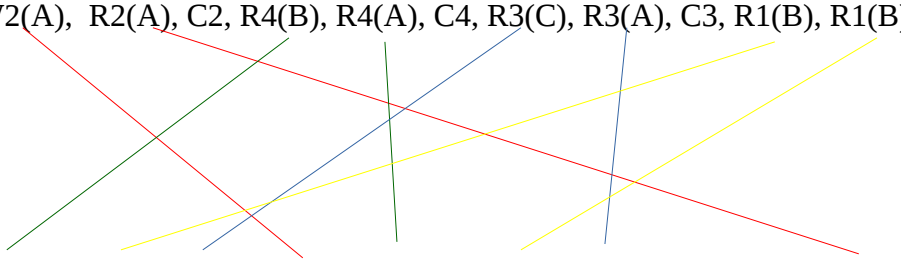
Assignment 5

Jyothi Prasad Nama Mahesh
N001783901

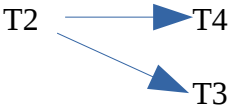
1. **R4(B), R1(B), R3(C), W2(A), R4(A), R1(B), R3(A), C3, C4, C1, R2(A), C2**

Conflicts for view serializability:
W2(A) R4(A) implies that T2 precedes T4
W2(A) R3(A) implies that T2 precedes T3
Consider the order T2, T4, T3, T1:
W2(A), R2(A), C2, R4(B), R4(A), C4, R3(C), R3(A), C3, R1(B), R1(B), C1



R4(B), R1(B), R3(C), W2(A), R4(A), R1(B), R3(A), C3, C4, C1, R2(A), C2
All read operations read the same value and all final values are the same => these sequences are View Equivalent
Conclusion: View Serializable in the order T2, T4, T3, T1

Conflicts for conflict serializability:
W2(A) R4(A)
W2(A) R3(A)
Transaction conflict graph:



Topological sort: T2, T4, T3
Conclusion: Conflict Serializable in the order T2, T4, T3, T1 or T1, T2, T4, T3…

There is a WR conflict
R4(B), R1(B), R3(C), W2(A), R4(A), R1(B), R3(A), C3, C4, C1, R2(A), C2
Conclusion: Not Recoverable, Not Cascadeless and Not Strict

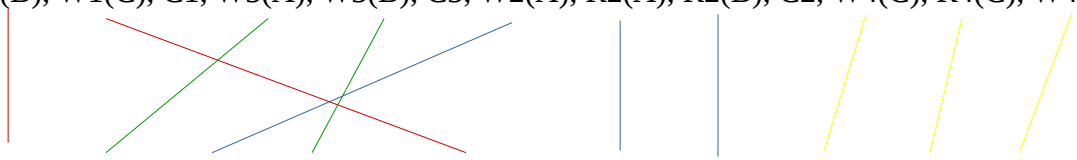
2. **R1(B), W3(A), W2(A), W3(B), C3, W1(C), C1, R2(A), R2(B), W4(C), R4(C), W4(C), C2, C4**

Conflicts for view serializability:

R1(B)	W3(B)	implies that T1 precedes T3
W3(A)	W2(A)	implies that T2 precedes T2
W3(B)	R2(B)	implies that T2 precedes T2
W1(C)	W4(C)	implies that T1 precedes T4

Consider the order T1, T3, T2, T4:

R1(B), W1(C), C1, W3(A), W3(B), C3, W2(A), R2(A), R2(B), C2, W4(C), R4(C), W4(C), C4



R1(B), W3(A), W2(A), W3(B), C3, W1(C), C1, R2(A), R2(B), W4(C), R4(C), W4(C), C2, C4

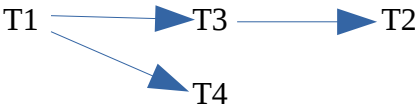
All read operations read the same value and all final values are the same => these sequences are View Equivalent

Conclusion: View Serializable in the order T1, T3, T2, T4

Conflicts for conflict serializability:

R1(B)	W3(B)
W3(A)	W2(A)
W3(B)	R2(B)
W1(C)	W4(C)

Transaction conflict graph:



Topological sort: T1, T3, T2, T4

Conclusion: Conflict Serializable in the order T1, T3, T2, T4 or T1, T3, T4, T2...

There are no WR conflicts so it is recoverable and cascadeless

There is a WW conflict

R1(B), W3(A), W2(A), W3(B), C3, W1(C), C1, R2(A), R2(B), W4(C), R4(C), W4(C), C2, C4

Conclusion: Recoverable and Cascadeless but Not Strict

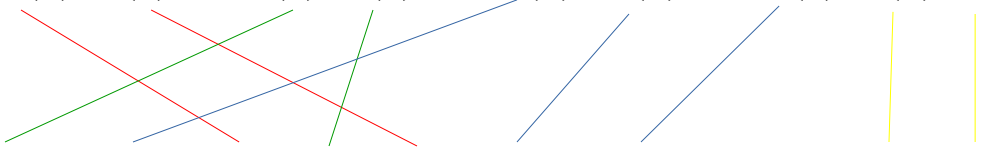
3. **R3(B), W2(A), W4(A), R3(B), R4(C), W2(A), W1(B), C2, C4, C3, R1(A), R1(C), C1**

Conflicts for view serializability:

R3(B)	W1(B)	implies that T3 precedes T1
W4(A)	W2(A)	implies that T4 precedes T2
R3(B)	W1(B)	implies that T3 precedes T1
W2(A)	R1(A)	implies that T2 precedes T1

Consider the order T4, T3, T2, T1:

W4(A), R4(C), C4, R3(B), R3(B), C3, W2(A), W2(A), C2, W1(B), R1(A), R1(C), C1



R3(B), W2(A), W4(A), R3(B), R4(C), W2(A), W1(B), C2, C4, C3, R1(A), R1(C), C1

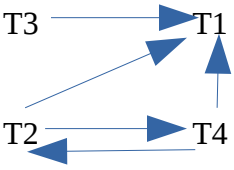
All read operations read the same value and all final values are the same => these sequences are View Equivalent

Conclusion: View Serializable in the order T4, T3, T2, T1

Conflicts for conflict serializability:

R3(B)	W1(B)
W4(A)	W2(A)
R3(B)	W1(B)
W2(A)	R1(A)
W2(A)	W4(A)
W4(A)	R1(A)

Transaction conflict graph:



Topological sort: Cannot to be topologically sorted because of the cycle between T2 and T4

Conclusion: Not Conflict Serializable

There are no WR conflicts so it is recoverable and cascadeless

There is a WW conflict

R3(B), W2(A), W4(A), R3(B), R4(C), W2(A), W1(B), C2, C4, C3, R1(A), R1(C), C1

Conclusion: Recoverable and Cascadeless but Not Strict

4. **R3(C), R2(A), W1(A), R1(B), R2(B), C2, R4(B), R4(C), W3(B), C4, C3, R1(B), R1(C), R1(B), C1**

Conflicts for view serializability:

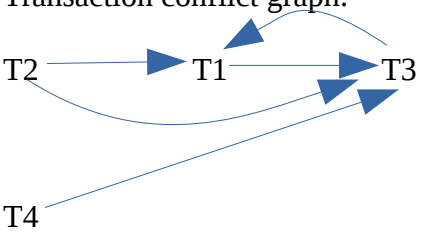
R2(A)	W1(A)	implies that T2 precedes T1
R1(B)	W3(B)	implies that T1 precedes T3
R2(B)	W3(B)	implies that T2 precedes T3
R4(B)	W3(B)	implies that T4 precedes T3
W3(B)	R1(B)	implies that T3 precedes T1

All permutations eliminated as it is not possible to choose any order with T1 preceding T3 and T3 preceding T1
Conclusion: Not View Serializable

Conflicts for conflict serializability:

R2(A)	W1(A)
R1(B)	W3(B)
R2(B)	W3(B)
R4(B)	W3(B)
W3(B)	R1(B)

Transaction conflict graph:



Topological sort: Cannot to be topologically sorted because of the cycle between T1 and T3
Conclusion: Not Conflict Serializable

There are no WR conflicts
There are no WW conflicts
Conclusion: Recoverable, Cascadeless and Strict

5. **R3(A), R1(A), W2(C), R2(A), R1(B), W2(A), R1(A), R3(B), W1(A), C1, C2, R3(C), R3(A), W3(C), R3(A), C3**

Conflicts for view serializability:

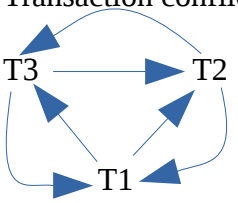
R3(A)	W2(A)	implies that T3 precedes T2
R1(A)	W2(A)	implies that T1 precedes T2
W2(C)	R3(C)	implies that T2 precedes T3

All permutations eliminated as it is not possible to choose any order with T3 preceding T2 and T2 preceding T3
Conclusion: Not View Serializable

Conflicts for conflict serializability:

R3(A)	W2(A)
R1(A)	W2(A)
W2(C)	R3(C)
W2(C)	W3(C)
W2(A)	R1(A)
W2(A)	W1(A)

Transaction conflict graph:

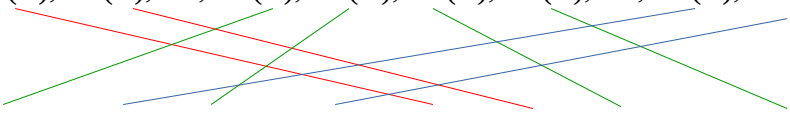


Topological sort: Cannot be topologically sorted because of all the cycles
Conclusion: Not Conflict Serializable

There is a WR conflict
R3(A), R1(A), W2(C), R2(A), R1(B), W2(A), R1(A), R3(B), W1(A), C1, C2, R3(C), R3(A), W3(C), R3(A), C3
Conclusion: Not Recoverable, Not Cascadeless and Not Strict

6. **R2(B), R3(B), R2(A), W3(A), C3, R1(C), R1(B), R2(B), C1, R2(C), C2**

Conflicts for view serializability:
R2(A) W3(A) implies that T2 precedes T3
Consider the order T1, T2, T3:
R1(C), R1(B), C1, R2(B), R2(A), R2(B), R2(C), C2, R3(B), W3(A), C3



R2(B), R3(B), R2(A), W3(A), C3, R1(C), R1(B), R2(B), C1, R2(C), C2
All read operations read the same value and all final values are the same => these sequences are View Equivalent
Conclusion: View Serializable in the order T1, T2, T3

Conflicts for conflict serializability:
R2(A) W3(A)
Transaction conflict graph:
T2 → T3
Topological sort: T2, T3, T1
Conclusion: Conflict Serializable in the order T2, T3, T1 or T1, T2, T3

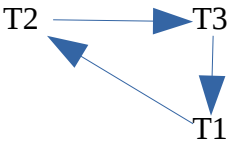
There are no WR conflicts
There are no WW conflicts
Conclusion: Recoverable, Cascadeless and Strict

---

7. **R2(A), R3(A), W1(B), W3(A), R1(A), R2(B), C1, R3(C), C3, C2**

Conflicts for view serializability:
R2(A) W3(A) implies that T2 precedes T3
W1(B) R2(B) implies that T1 precedes T2
W3(A) R1(A) implies that T3 precedes T1
All permutations eliminated as it is not possible to choose any order with T2 preceding T3, T1 preceding T2 and T3 preceding T1
Conclusion: Not View Serializable

Conflicts for conflict serializability:
R2(A) W3(A)
W1(B) R2(B)
W3(A) R1(A)
Transaction conflict graph:



Topological sort: Cannot be topologically sorted because of the cycle formed by T2 → T3 → T1 → T2
Conclusion: Not Conflict Serializable

There is a WR conflict
R2(A), R3(A), W1(B), W3(A), R1(A), R2(B), C1, R3(C), C3, C2
Conclusion: Not Recoverable, Not Cascadeless and Not Strict

---

8. **R1(B), W2(C), R1(C), C1, W3(A), R2(A), R2(C), R3(A), W3(A), R3(B), W3(A), W3(C), R2(B), C2, W3(C), R3(B), C3**

Conflicts for view serializability:

W2(C)	R1(C)	implies that T2 precedes T1
W2(C)	W3(C)	implies that T2 precedes T3
W3(A)	R2(A)	implies that T3 precedes T2

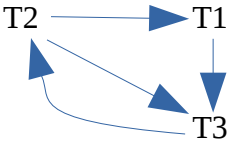
All permutations eliminated as it is not possible to choose any order where T2 precedes T3 and T3 precedes T2

Conclusion: Not View Serializable

Conflicts for conflict serializability:

W2(C)	R1(C)
W2(C)	W3(C)
W3(A)	R2(A)
R1(C)	W3(C)

Transaction conflict graph:



Topological sort: Cannot be topologically sorted because of the cycles between T2 and T3

Conclusion: Not Conflict Serializable

There is a WR conflict

R1(B), W2(C), R1(C), C1, W3(A), R2(A), R2(C), R3(A), W3(A), R3(B), W3(A), W3(C), R2(B), C2, W3(C), R3(B), C3

Conclusion: Not Recoverable, Not Cascadeless and Not Strict

---

a. Show the name of the employee's spouse, if the employee has a spouse; otherwise show "unmarried".
XPath 1.0

```
concat (    substring(/Employee/Dependent[@relatedBy="Spouse"]/text(), 0),
            substring("unmarried", 10*boolean(/Employee/Dependent[@relatedBy="Spouse"])) )
```

HAPL JSON Semantics

```
concat (    substring(//Employee/Dependent/*[relatedBy="Spouse"]/name, 0),
            substring("unmarried", 9*boolean(//Employee/Dependent/*[relatedBy="Spouse"])))
```

Reference: http://stackoverflow.com/questions/7045151/return-a-string-value-based-on-xpath-condition

b. List all the benefits with weekly employee cost greater than 20.
XPath 1.0

```
//Benefit[Plan/Cost[@period="week" and EmployeeCost>20]]
```

HAPL JSON Semantics

```
//Benefit/*[Plan/Cost/*/period="week" and Plan/Cost/*/EmployeeCost>20]
```

c. Find the address of the professor who has at least three dependents.
XPath 1.0

```
/Employee[Position="Professor" and count(Dependent)>=3]/Address
```

HAPL JSON Semantics

```
//Employee[Position="Professor" and count(Dependent/*)>=3]/Address
```

d. List the plan name of all benefits with employee cost less than twice the subsidy given by the employer on a weekly basis.
XPath 1.0

```
//Benefit/Plan[Cost[EmployeeCost<2*EmployeeSubsidy and @period="week"]]/PlanName
```

HAPL JSON Semantics

```
//Benefit/*/Plan[Cost/*[period="week" and EmployeeCost < 2*EmployerSubsidy]]/PlanName
```

e. Find the employee whose address ends in "Boston, MA" and whose spouse's name contains "Andesi" in it.
XPath 1.0

```
/Employee[ends-with(Address,"Boston, MA") and Dependent[@relatedBy="Spouse" and
contains(.,"Andesi")]]
```

HAPL JSON Semantics

```
//Employee[substring(Address, string-length(Address)-10)="Boston, MA" and
Dependent/*[relatedBy="Spouse" and contains(name,"Andesi")]]
```

f. List the name of the employee who named all his children with a first name that starts with the letter 'H'.
XPath 1.0

```
/Employee[count(Dependent[@relatedBy="Child" and starts-
with(.,"H")])=count(Dependent[@relatedBy="Child"])]/Name
```

HAPL JSON Semantics

```
//Employee[count(Dependent/*[relatedBy="Child" and starts-
with(name,"H")])=count(Dependent/*[relatedBy="Child"])]/Name
```