**CS 411 Database Systems Homework 4**

**Spring 2023**

**Submitted by: Payal Mantri (payaljm2)**

1. **R1(A); R1(B); W1(B); R1(C); W1(C); W1(D); R1(E); W1(E)**Explanation:  
   T1 reads mobile phone specification and preorder information from A and B. It also writes a preorder request to B.  
     
   T1 reads delivery data from C and writes selected details in C  
     
   T1 generates the bill and writes to element D

T1 fetches list E. Customer bill, phone number, is added to this list and stored back to E

1. **Isolation Level and Locking**
   1. **T1: Repeatable Read T2: Read Committed**

**S1: R1(A), R2(C), R1(B), W1(C), R2(A), W2(B), R1(A), R1(B), R2(A)  
S2: R2(A), R1(B), R2(C), R1(A), W1(C), R2(C), W2(B), R2(A)**

**S1:** SLOCK1(A); R1(A); SLOCK2(C); R2(C); REL2(C); SLOCK1(B); R1(B); XLOCK1(C); W1(C); SLOCK2(A); R2(A); REL2(A); XLOCK2(B); DENIED2(B);   
  
The lock table is as shown .

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| SLOCK1 | SLOCK1 | ~~SLOCK2~~ |
| SLOCK2 | XLOCK2 - DENIED | XLOCK2 |

T2 tries acquiring exclusive lock on B , but T1 already holds a shared lock on B.And since T1 is on repeatable read isolation level, it wont release this shared lock until T1 is completed . Hence T2 is denied the lock and Schedule **S1 is not feasible under** given isolation levels

**S2:** SLOCK2(A); R2(A); REL2(A); SLOCK1(B); R1(B); SLOCK2(C); R2(C); REL2(C); SLOCK1(A); R1(A); XLOCK1(C); W1(C); REL1(A, B, C); SLOCK2(C); R2(C); REL2(C); XLOCK2(B); W2(B); SLOCK2(A); R2(A); REL2(A, B);

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| ~~SLOCK2~~ | ~~SLOCK1~~ | ~~SLOCK2~~ |
| ~~SLOCK1~~ | ~~XLOCK2~~ | ~~XLOCK1~~ |
| ~~SLOCK2~~ |  | ~~SLOCK2~~ |

Schedule S**2 is feasible under** given isolation levels

* 1. **T1: Repeatable Read T2: Repeatable Read**

**S1:** *SLOCK1(A); R1(A); SLOCK2(C); R2(C); SLOCK1(B); R1(B); XLOCK1(C); DENIED1(C);*

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| SLOCK1 | SLOCK1 | SLOCK2 |
|  |  | XLOCK1 - DENIED |

T1 is Denied exclusive lock on C as there already exists a shared lock on C by T2 which will be released only when T2 completes. Hence this schedule **S1 is infeasible** under given isolation levels.

**S2:** *SLOCK2(A); R2(A); SLOCK1(B); R1(B); SLOCK2(C); R2(C); SLOCK1(A); R1(A); XLOCK1(C); DENIED1(C);*

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| SLOCK2 | SLOCK1 | SLOCK2 |
| SLOCK1 |  | XLOCK1 - DENIED |

T1 is Denied exclusive lock on C as there already exists a shared lock on C by T2 which will be released only when T2 completes. Hence this schedule **S2 is infeasible** under given isolation levels

* 1. **T1: Repeatable Read; T2: Repeatable Read**

**S1: Q1(T2), Q2(T1), Commit(T1), Q1(T2)**

|  |  |
| --- | --- |
| Transaction1 T1 | Transaction2 T2 |
| START TRANSACTION | START TRANSACTION |
|  | **Q1(T2) \*acquires shared lock on 2 rows SLOCK(tempId=1) SLOCK(tempId = 2000)  SELECT \* FROM Temperatures WHERE cityName=’Portland’;** /reads 2 rows/  **Result:**   |  |  |  | | --- | --- | --- | | tempId | **temperature** | **cityName** | | **1** | **50** | **Portland** | | **2000** | **90** | **Portland** | |
| Q2(T1) \*Acquires exclusive lock on new row with id =10001 XLOCK(tempId=100041)  INSERT INTO Temperatures (tempId, temperature, cityName) VALUES (10001, 60, ‘Portland’); |  |
| Commit(T1) COMMIT;  \*Releases exclusive lock on new row with id =10001 REL1(tempID=100041) |  |
|  | **Q1(T2)**  **Acquires new shared lock on row with id 100041 SLOCK(tempId=100041)  SELECT \* FROM Temperatures WHERE cityName=’Portland’;** /reads 3 rows/  **Result:**   |  |  |  | | --- | --- | --- | | tempId | **temperature** | **cityName** | | **1** | **50** | **Portland** | | **2000** | **90** | **Portland** | | **10001** | **60** | **Portland** |   **and this new row was added by T1 , hence now T2 sees 2 results instead of 1 , thus leading to phantom reads.  REL(tempId =2 , tempId=2000, tempId=100041)** |

We can see phantom reads in Transaction 2 as the first select query results in 2 row s but later the same query gives 3 rows as output. This happens because transaction T1 inserts a new row in between. Hence under given isolation settings, we can see phantom reads

* 1. **S2: Q1(T2), Q2(T1), Q1(T2), Abort(T1)  
     T1: Read Committed; T2: Read Uncommitted.**

|  |  |
| --- | --- |
| Transaction1 T1 | Transaction2 T2 |
| START TRANSACTION | START TRANSACTION |
|  | **\*acquires shared lock on 2 rows SLOCK(tempId=1) SLOCK(tempId = 2000)  SELECT \* FROM Temperatures WHERE cityName=’Portland’;** /reads 2 rows/  **Result:**   |  |  |  | | --- | --- | --- | | tempId | **temperature** | **cityName** | | **1** | **50** | **Portland** | | **2000** | **90** | **Portland** | |
| \*Acquires exclusive lock on new row with id =10001 XLOCK(tempId=100041)  INSERT INTO Temperatures (tempId, temperature, cityName) VALUES (10001, 60, ‘Portland’); |  |
|  | **Acquires new shared lock on row with id 100041 SLOCK(tempId=100041)  SELECT \* FROM Temperatures WHERE cityName=’Portland’;** /reads 3 rows/  **Result:**   |  |  |  | | --- | --- | --- | | tempId | **temperature** | **cityName** | | **1** | **50** | **Portland** | | **2000** | **90** | **Portland** | | **10001** | **60** | **Portland** |   **and this new row was added by T1. Even though this change is not commit, but T2 has read uncommitted setting hence now T2 sees 3 results instead of 2 , thus leading to dirty reads** |
| ABORT REL(tempId = 100041) |  |
|  | **REL(tempId =2 , tempId=2000, tempId=100041)** |

The transaction T2 sees dirty read problem. Initially, T2 sees 2 rows based on cityName=Portland and in the same time T! inserts a new row with same value. NowT2 runs query again to see 3 rows as it can read uncommitted data based on its isolation level of Read uncommitted. This is what is called dirty read where transaction reads a value written by another transaction that has not yet been committed

1. 1. **S1: R1(A); R1(B); W1(B); R2(B); W2(B); R1(C); W1(C); R2(C); W2(C); R2(A); R3(A); W3(A);   
        
      Solution:**  The schedule can be written in tabular form as

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| R(A) |  |  |
| R(B) |  |  |
| W(B) |  |  |
|  | R(B) |  |
|  | W(B) |  |
| R(C) |  |  |
| W(C) |  |  |
|  | R(C) |  |
|  | W(C) |  |
|  | R(A) |  |
|  |  | R(A) |
|  |  | W(A) |

The precedence graph is as shown below. Since the graph has no cycles, the schedule is conflict-serializable **Diagram

Description automatically generated**

The equivalent Serial schedule is   
  
**R1(A); R1(B); W1(B); R2(B); W2(B); R1(C); W1(C); R2(C); W2(C); R2(A); R3(A); W3(A);   
  
R1(A); R1(B); W1(B); R2(B); R1(C);W2(B); W1(C); R2(C); W2(C); R2(A); R3(A); W3(A);**

**R1(A); R1(B); W1(B); R1(C); R2(B);W2(B); W1(C); R2(C); W2(C); R2(A); R3(A); W3(A);  
  
R1(A); R1(B); W1(B); R1(C); R2(B);W1(C); W2(B); R2(C); W2(C); R2(A); R3(A); W3(A);  
  
Final Serial Schedule  
R1(A); R1(B); W1(B); R1(C); W1(C);R2(B); W2(B); R2(C); W2(C); R2(A); R3(A); W3(A);**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| R(A) |  |  |
| R(B) |  |  |
| W(B) |  |  |
| R(C) |  |  |
| W(C) |  |  |
|  | R(B) |  |
|  | W(B) |  |
|  | R(C) |  |
|  | W(C) |  |
|  | R(A) |  |
|  |  | R(A) |
|  |  | W(A) |

* 1. **S2: R3(C); W1(B); W2(B); W3(A); R1(C); W2(A); W1(A);**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
|  |  | R(C) |
| W(B) |  |  |
|  | W(B) |  |
|  |  | W(A) |
| R(C) |  |  |
|  | W(A) |  |
| W(A) |  |  |
|  |  |  |

Diagram

Description automatically generated

There is a cycle in the precedence graph and hence the schedule **S2 is not conflict-serializable.**

The transactions that make the schedule not conflict-serializable are **T1 and T2 .**  
The actions involved are   
**T1 T2 : W1(B);W2(B); and   
T2 T1: W2(A);W1(A);**

1. **2PL locking** 
   1. **S1: W3(C), W1(A), R2(B), W3(B), R2(B), W1(B)  
      S2: W1(C); W3(B); R3(B); W2(A); R1(A); R3(A); R2(A); R2(C); R2(B)**   
      **S3: R1(A); R3(B); R2(C); R1(B); R3(C); W1(A); R1(A); R3(B); W2(C)**

**S1: XLOCK3(C); W3(C); XLOCK1(A); W1(A); SLOCK2(B); R2(B); XLOCK 3(B):DENIED3(B)**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
|  |  | XLOCK3(C) |
|  |  | W3(C) |
| XLOCK1(A) |  |  |
| W1(A) |  |  |
|  | SLOCK2(B) |  |
|  | R2(B) |  |
|  |  | XLOCK3(B) <Denied, Wait> |

Since transaction 3 is denied exclusive lock for B as T2 is holding shared lock on b**. Schedule S1 cannot arise from 2PL**

**S2: W1(C); W3(B); R3(B); W2(A); R1(A); R3(A); R2(A); R2(C); R2(B)**

**S2: XLOCK1(C); W1(C); XLOCK3(B); W3(B); R3(B); XLOCK2(A); W2(A); SLOCK1(A); DENIED1(A);**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| XLOCK1(C) |  |  |
| W1(C) |  |  |
|  |  | XLOCK3(B) |
|  |  | W3(B) |
|  |  | R3(B) |
|  | XLOCK2(A) |  |
|  | W2(A) |  |
| SLOCK1(A) <Denied, Wait> |  |  |

**Schedule S2 cannot arise from 2PL scheduler**

**S3: R1(A); R3(B); R2(C); R1(B); R3(C); W1(A); R1(A); R3(B); W2(C)**

**S3: SLOCK1(A); R1(A); SLOCK3(B); R3(B); SLOCK2(C); R2(C); SLOCK1(B); R1(B); SLOCK3(C); R3(C); REL3(C); XLOCK1(A); W1(A); REL1(B); SLOCK1(A); R1(A); COMMIT1(A); REL1(A); R3(B); COMMIT(T3); REL3(B); XLOCK2(C); W2(C); COMMIT(T3); REL2(C)**Assumption:  In case we need to release some locks and downgrade others, release locks FIRST, then downgrade the others.

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| SLOCK1(A) |  |  |
| R1(A) |  |  |
|  |  | SLOCK3(B) |
|  |  | R3(B) |
|  | SLOCK2(C) |  |
|  | R2(C) |  |
| SLOCK1(B) |  |  |
| R1(B) |  |  |
|  |  | SLOCK3(C) |
|  |  | R3(C) |
|  |  | REL3(C) |
| XLOCK1(A) |  |  |
| W1(A) |  |  |
| REL1( B) |  |  |
| SLOCK1(A) |  |  |
| R1(A) |  |  |
| COMMIT(T1) |  |  |
| REL1(A) |  |  |
|  |  | R3(B) |
|  |  | COMMIT(T3) |
|  |  | REL3(B) |
|  | XLOCK2(C) |  |
|  | W2(C) |  |
|  | COMMIT(T2) |  |
|  | REL2(C) |  |

The Schedule S3 can be produced by a Two-Phase Lock (2PL) scheduler.

* 1. **S1: W3(C), W1(A), R2(B), W3(B), R2(B), W1(B)  
       
     XLOCK3(C); W3(C); XLOCK1(A); W1(A); SLOCK2(B); R2(B); XLOCK3(B); DENIED3(B)  
       
     S1 is not feasible using strict 2PL . We can enforce it to obtain following schedule  
       
     Enforcing String 2PL:-   
       
     S1: XLOCK3(C); W3(C); XLOCK1(A); W1(A); SLOCK2(B); R2(B); XLOCK3(B); WAIT3(B); R2(B); COMMIT T2; REL2(B); XLOCK3(B); W3(B); COMMIT T3; REL3(B, C); XLOCK1(B); W1(B); REL1(A,B); COMMIT T1;**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
|  |  | XLOCK3(C) |
|  |  | W3(C) |
| XLOCK1(A) |  |  |
| W1(A) |  |  |
|  | SLOCK2(B) |  |
|  | R2(B) |  |
|  |  | XLOCK3(B) <Denied, Wait> |
|  | R2(B) |  |
|  | COMMIT; |  |
|  | REL2(B) |  |
|  |  | XLOCK3(B) |
|  |  | W3(B) |
|  |  | COMMIT |
|  |  | REL3(B, C) |
| XLOCK1(B) |  |  |
| W1(B) |  |  |
| COMMIT; |  |  |
| REL1(A, B) |  |  |

**S2: W1(C); W3(B); R3(B); W2(A); R1(A); R3(A); R2(A); R2(C); R2(B)  
  
XLOCK1(C); W1(C); XLOCK3(B); W3(B); R3(B); XLOCK2(A); W2(A); SLOCK1(A); DENIED1(A);**

**S2 is not feasible using strict 2 PL.  
  
Enforcing Strict 2PL on S2: -  
  
S2: XLOCK1(C); W1(C); XLOCK3(B); W3(B); R3(B); XLOCK2(A); W2(A); SLOCK1(A); WAIT1(A); SLOCK3(A); WAIT3(A); R2(A); SLOCK2(C); WAIT2(C) ;**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| XLOCK1(C) |  |  |
| W1(C) |  |  |
|  |  | XLOCK3(B) |
|  |  | W3(B) |
|  |  | R3(B) |
|  | XLOCK2(A) |  |
|  | W2(A) |  |
| SLOCK1(A) <Denied, Wait> |  |  |
|  |  | SLOCK3(A)< WAIT> |
|  | R2(A) |  |
|  | SLOCK2(C)< WAIT> |  |

T1 is waiting for Shared lock A. It can be obtained only when T2 finishes.  
T2 is waiting on Shared lock C, which can be obtained when T1 completes. **Hence there is a deadlock**

**S3: R1(A); R3(B); R2(C); R1(B); R3(C); W1(A); R1(A); R3(B); W2(C)**

**S3: SOCK1(A); R1(A); SLOCK3(B); R3(B); SLOCK2(C); R2(C); SLOCK1(B); R1(B); SLOCK3(C); R3(C); XLOCK1(A); W1(A); R1(A); COMMIT T1; REL1(A, B); R3(B); COMMIT T3; REL3(B,C); XLOCK2(C); W2(C); COMMIT T2; REL2(C);**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| SLOCK1(A) |  |  |
| R1(A) |  |  |
|  |  | SLOCK3(B) |
|  |  | R3(B) |
|  | SLOCK2(C) |  |
|  | R2(C) |  |
| SLOCK1(B) |  |  |
| R1(B) |  |  |
|  |  | SLOCK3(C) |
|  |  | R3(C) |
| XLOCK1(A) |  |  |
| W1(A) |  |  |
| R1(A) |  |  |
| COMMIT(T1) |  |  |
| REL1(A, B) |  |  |
|  |  | R3(B) |
|  |  | COMMIT(T3) |
|  |  | REL3(B, C) |
|  | XLOCK2(C) |  |
|  | W2(C) |  |
|  | COMMIT(T2) |  |
|  | REL2(C) |  |

Hence the **Schedule S3 can be produced by a Strict Two-Phase Lock (2PL)** scheduler.