# Student ID: 201447766

# COMP207 Assignment 1 – Transaction Management

**Issue Date:** Thursday, 31 October 2019

**Submission Deadline:** Thursday, 7 November 2019, 17:00

# About This Assignment

This is the first of two assignments for COMP207. It is worth 10% of the total marks for this module. It consists of six questions, which you can find at the end of this document.

Submit your solutions to these questions in PDF format by the given submission deadline. Your solutions must be submitted on Vital (see the detailed submission instructions below).

Accuracy and relevance are more important in your answers, so don’t write large volumes in your submission, but do ensure that what you write covers what is asked for and keeps to the problem statement.

# Submission Details

Please submit **one PDF file with your solutions**. Name your file as follows:

<your student ID>-Assignment-1.pdf

If your student ID is 12345678, then your file should be named: 12345678-Assignment-1.pdf.

Please submit only this file (no archives).

To act as your ‘signature’ for the assignment, at the top of your PDF document put your

Student ID number.

Your solutions must be submitted on Vital (see Vital for submission instructions).

The submission deadline for this assignment is **Thursday, 7 November 2019, 17:00**. Earlier submission is possible, but any submission after the deadline attracts the standard lateness penalties. Plagiarism and collusion guidelines will apply throughout the assignment submission. For details on late submissions, how to claim extenuating circumstances, etc., please see the undergraduate student handbook, which can be found at <http://intranet.csc.liv.ac.uk/student/ug-handbook.pdf>, or in Section 6 of the Code of Practice on Assessment.1

# Assessment information at a glance

**Assignment Number** 1 (of 2)

**Weighting** 10% of the final module mark

**Assignment Circulated** Thursday, 31 October 2019

**Deadline** Thursday, 7 November 2019, 17:00

**Submission Mode** Electronically on Vital

LO1: Identify and apply the principles underpinning transaction management within DBMS and the main security issues involved in securing transactions

**Learning Outcome Assessed**

**Purpose of Assessment** Assessment of knowledge of multi-user databases and the

need for relevant control to ensure database integrity

**Marking Criteria** See description of this assignment

### Submission necessary in N/A

**order to satisfy module requirements?**

**Late Submission Penalty** Standard UoL Policy

1 [https://www.liverpool.ac.uk/media/livacuk/tqsd/code-of-practice-on-](https://www.liverpool.ac.uk/media/livacuk/tqsd/code-of-practice-on-assessment/code_of_practice_on_assessment.pdf) [assessment/code\_of\_practice\_on\_assessment.pdf](https://www.liverpool.ac.uk/media/livacuk/tqsd/code-of-practice-on-assessment/code_of_practice_on_assessment.pdf)

**Question 1 (16 marks)**

Consider the following two transactions (we omit the final ‘commit’ operation):

### Transaction *T*1

read item(*X* );

*X* := *X* + 2;

write item(*X* ); read item(*Y* ); *Y* := *Y* \* 3;

write item(*Y* );

commit;

### Transaction *T*2

read item(*Y* );

*Y* := *Y* \* 2;

write item(*Y* );

read item(*X* );

*X* := *X* + 3;

write item(*X* ); commit;

Assume that transactions *T*1 and *T*2 use shared buffers (i.e., once a transaction writes *X* back to the buffer, the new value of *X* can be read by the other transaction, and similarly for *Y* ).

1. Give serial schedules *S*1 for *T*1  *T*2 and *S*2 for *T*2  *T*1. (1 mark per schedule)

Ans:

S1: r­1(X); w1(X); r1(Y); w1(Y); r­2(Y); w2(Y); r­2(X); w2(X);

S2: r­2(Y); w2(Y); r­2(X); w2(X); r­1(X); w1(X); r1(Y); w1(Y);

1. For each of the two schedules you gave in **(a)** *S*1 and *S*2, give the values of *X* and *Y*

after executing the schedule on a database with items *X* = 1 and *Y* = 2?

(2 marks per schedule)

Ans:

In S1, X=6 and Y=12;

In S2, X=6 and Y=12.

1. Give a serialisable schedule *S*3 for *T*1 and *T*2 that is not serial. Explain why your schedule *S*3 is serialisable. (5 marks)

Ans:

S3: r­1(X); w1(X); r­2(Y); w2(Y); r­2(X); w2(X); r1(Y); w1(Y);

Explanation: S1 and S2 in (a) has the same effect as S3 on every initial database state. Since S1 is a serial schedule, S3 is serializable.

1. Give a schedule *S*4 for *T*1 and *T*2 that is not serialisable. Explain why *S*4 is not serialisable. (5 marks)

Ans:

S4: r­1(X); w1(X); r­2(Y); r1(Y); w1(Y); w2(Y); r­2(X); w2(X);

Explanation:

After executing the schedule S4 on database, X=X+6 but Y=3\*Y. Therefore, it means that S4 does not have the same effect as serial schedules *S*1 or S2. Thus, is not serialisable.

## Question 2 (35 marks)

Consider the following schedules:

* + *S*1 : *r*1(*X* ); *r*2(*X* ); *r*3(*Y* ); *w*1(*X* ); *r*2(*Z*); *r*2(*Y* ); *w*2(*Y* ); *w*1(*Z*)
  + *S*2 : *r*1(*X* ); *r*1(*Y* ); *w*1(*Y* ); *r*3(*Y* ); *r*2(*Y* ); *r*2(*Z*); *w*3(*U* ); *w*2(*Z*); *r*4(*Z*); *r*4(*U* ); *w*4(*U* )
  + *S*3 : *w*3(*X* ); *r*1(*X* ); *w*1(*Y* ); *r*2(*Y* ); *w*2(*Z*); *r*3(*Z*)
  + *S*4 : *r*1(*X* ); *r*4(*X* ); *r*3(*Y* ); *w*4(*Y* ); *r*1(*Y* ); *r*2(*Y* ); *r*3(*X* ); *r*4(*Y* ); *w*1(*X* ); *w*2(*Y* )
  + *S*5 : *r*1(*X* ); *w*1(*Y* ); *r*2(*Y* ); *w*2(*Z*); *r*3(*Z*); *w*3(*X* )

For each of these schedules, answer the following questions:

1. What are the conflicts in the schedule? (2 marks per schedule)

Ans:

Conflict pairs in these schedules:

S1 :*r*2(*X* )-*w*1(*X* ) , *r*3(*Y* )-*w*2(*Y* ) , *r*2(*Z*)- *w*1(*Z*)

*S*2 : *w*1(*Y* )-*r*2(*Y* ) , *w*1(*Y* )-*r*3(*Y* ) , *w*3(*U* )-*w*4(*U* ) , *w*3(*U* )-*r*4(*U* ) , *w*2(*Z*)-*r*4(*Z*)

*S*3 : *w*3(*X* )-*r*1(*X* ) , *w*1(*Y* )-*r*2(*Y* ) , *w*2(*Z*)- *r*3(*Z*)

*S*4 : *r*3(*Y* )-*w*4(*Y* ) , *w*4(*Y* )-*r*1(*Y* ) , *w*4(*Y* )-*r*2(*Y* ) , *w*4(*Y* )- *w*2(*Y* ) , *r*4(*X* )- *w*1(*X* ) , *r*3(*X* )- *w*1(*X* ) , *r*3(*Y* )- *w*2(*Y* ) , *r*1(*Y* )- *w*2(*Y* ) , *r*4(*Y* )-*w*2(*Y* )

*S*5 : *w*1(*Y* )- *r*2(*Y* ) , *w*2(*Z*)-*r*3(*Z*) , *r*1(*X* )-*w*3(*X* )

1. What is the precedence graph of the schedule? (2 marks per schedule)

Ans:

S1 :

T1

T2

T3

*S*2 :

T1

T2

T3

T4

*S*3 :

T1

T2

T3

*S*4 :

T4

T3

T2

T1

*S*5 :

T1

T2

T3

1. Is the schedule conflict-serialisable? Why? If the schedule is conflict-serialisable, give a conflict-equivalent serial schedule. (3 marks per schedule)

Ans:

S1: S1 is conflict-serialisable because it has no cycle in its precedence graph.

conflict-equivalent serial schedule of S1:

*r*3(*Y* ); *r*2(*X* ); *r*2(*Z*); *r*2(*Y* ); *w*2(*Y* ); *r*1(*X* ); *w*1(*X* ); *w*1(*Z*)

S2: S2 is conflict-serialisable because it has no cycle in its precedence graph.

conflict-equivalent serial schedule of S2:

*r*1(*X* ); *r*1(*Y* ); *w*1(*Y* ); *r*2(*Y* ); *r*2(*Z*); *w*2(*Z*); *r*3(*Y* ); *w*3(*U* ); *r*4(*Z*); *r*4(*U* ); *w*4(*U* )

S3: S3 is not conflict-serialisable because it has a cycle in its precedence graph.

S4: S4 is conflict-serialisable because it has no cycle in its precedence graph.

conflict-equivalent serial schedule of S4:

*r*3(*Y* ); *r*3(*X* ); *r*4(*X* ); *w*4(*Y* ); *r*4(*Y* ); *r*1(*X* ); *r*1(*Y* ); *w*1(*X* ); *r*2(*Y* ); *w*2(*Y* )

S5: S5 is conflict-serialisable because it has no cycle in its precedence graph.

conflict-equivalent serial schedule of S5:

*r*1(*X* ); *w*1(*Y* ); *r*2(*Y* ); *w*2(*Z*); *r*3(*Z*); *w*3(*X* )

## Question 3 (15 marks)

For each of the following schedules, determine if the schedule is:

1. Recoverable
2. Cascadeless
3. Strict

**(a)** *S*1 : *r*1(*X* ); *r*2(*X* ); *w*2(*X* ); *w*1(*X*); *a*2*; c*1 (3 marks)

Recoverable

Cascadeless

Not strict

**(b)** *S*2 : *r*1(*X* ); *r*2(*X* ); *w*2(*X* ); *w*1(*X*); *c*2*; c*1 (3 marks)

Recoverable

Cascadeless

Not strict

**(c)** *S*3 : *r*1(*X* ); *w*1(*X* ); *r*2(*X* ); *w*1(*X*); *c*2*; c*1 (3 marks)

Not recoverable

Not cascadeless

Not strict

**(d)** *S*4 : *r*1(*X* ); *w*1(*X* ); *r*2(*X* ); *w*1(*X*); *a*2*; c*1 (3 marks)

Recoverable

Not cascadeless

Not strict

**(e)** *S*5 : *r*1(*X* ); *r*2(*X* ); *w*2(*X* ); *c*2*; w*1(*X*); *c*1*; r*3(*X*); *c*3 (3 marks)

Recoverable

Cascadeless

Strict

## Question 4 (18 marks)

For each of the following sequences of operations, simulate its execution until it finishes or cannot proceed. If the execution cannot proceed, explain why.

For each step, indicate which of the two transactions *T*1 and *T*2 holds which type of lock on *X*

and *Y*.

**Note.** Each schedule spans two lines of text.

**(a)** *S*1 : *sl*1(*X* ); *r*1(*X* ); *ul*1(*Y* ); *r*1(*Y* ); *sl*2(*Y* ); *r*2(*Y* ); *sl*2(*X* ); *r*2(*X* ); *u*2(*X* ); *u*2(*Y* );

*xl*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *u*1(*X* ) (6 marks)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | T1 | T2 | Lock status | |
| Locks owned by T1 | Locks owned by T2 |
| *1* | *s-lock*(*X*) |  | *s-lock*(*X* ) |  |
| *2* | *read\_item(X)* |  | *s-lock*(*X* ) |  |
| *3* | *u-lock(Y)* |  | *s-lock*(*X* ), *u-lock(Y)* |  |
| *4* | *read\_item(Y)* |  | *s-lock*(*X* ), *u-lock(Y)* |  |
| 5 |  | *s-lock(Y)*  denied | *s-lock*(*X* ), *u-lock(Y)* |  |
| 6 |  | *s-lock*(*X*) | *s-lock*(*X* ), *u-lock(Y)* | *s-lock*(*X*) |
| 7 |  | *read\_item(X)* | *s-lock*(*X* ), *u-lock(Y)* | *s-lock*(*X*) |
| 8 |  | *unlock(X)* | *s-lock*(*X* ), *u-lock(Y)* |  |
| 9 | *x-lock*(*Y*) |  | *s-lock*(*X* ), *u-lock(Y), x-lock*(*Y*) |  |
| 10 | *write\_item(Y)* |  | *s-lock*(*X* ), *u-lock(Y), x-lock*(*Y*) |  |
| 11 | *unlock(Y)* |  | *s-lock*(*X* ) |  |
| 12 |  | *s-lock(Y)* | *s-lock*(*X* ) | *s-lock(Y)* |
| 13 |  | *read\_item(Y)* | *s-lock*(*X* ) | *s-lock(Y)* |
| 14 |  | *unlock(Y)* | *s-lock*(*X* ) |  |
| 15 | *unlock(X)* |  |  |  |

**(b)** *S*2 : *sl*1(*X* ); *r*1(*X* ); *sl*2(*Y* ); *r*2(*Y* ); *ul*1(*Y* ); *r*1(*Y* ); *sl*2(*X* ); *r*2(*X* ); *u*2(*X* ); *u*2(*Y* );

*xl*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *u*1(*X* ) (6 marks)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | T1 | T2 | Lock status | |
| Locks owned by T1 | Locks owned by T2 |
| *1* | *s-lock*(*X*) |  | *s-lock*(*X* ) |  |
| *2* | *read\_item(X)* |  | *s-lock*(*X* ) |  |
| *3* |  | *s-lock*(*Y*) | *s-lock*(*X* ) | *s-lock*(*Y*) |
| *4* |  | *read\_item(Y)* | *s-lock*(*X* ) | *s-lock*(*Y*) |
| 5 | *u-lock*(*Y*) |  | *s-lock*(*X* ), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 6 | *read\_item(Y)* |  | *s-lock*(*X* ), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 7 |  | *s-lock*(*X*) | *s-lock*(*X* ), *u-lock*(*Y*) | *s-lock*(*Y*), *s-lock*(*X*) |
| 8 |  | *read\_item(X)* | *s-lock*(*X* ), *u-lock*(*Y*) | *s-lock*(*Y*), *s-lock*(*X*) |
| 9 |  | *unlock(X)* | *s-lock*(*X* ), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 10 |  | *unlock(Y)* | *s-lock*(*X* ), *u-lock*(*Y*) |  |
| 11 | *x-lock*(*Y*) |  | *s-lock*(*X* ), *u-lock*(*Y*), *x-lock*(*Y*) |  |
| 12 | *write\_item(Y)* |  | *s-lock*(*X* ), *u-lock*(*Y*), *x-lock*(*Y*) |  |
| 13 | *unlock(Y)* |  | *s-lock*(*X* ) |  |
| 14 | *unlock(X)* |  |  |  |

**(c)** *S*3 : *sl*2(*Y* ); *r*2(*Y* ); *sl*1(*X* ); *r*1(*X* ); *ul*1(*Y* ); *r*1(*Y* ); *xl*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *u*1(*X* );

*sl*2(*X* ); *r*2(*X* ); *u*2(*X* ); *u*2(*Y* ) (6 marks)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | T1 | T2 | Lock status | |
| Locks owned by T1 | Locks owned by T2 |
| *1* |  | *s-lock*(*Y*) |  | *s-lock*(*Y*) |
| *2* |  | *read\_item(Y)* |  | *s-lock*(*Y*) |
| *3* | *s-lock*(*X*) |  | *s-lock*(*X*) | *s-lock*(*Y*) |
| *4* | *read\_item(X)* |  | *s-lock*(*X*) | *s-lock*(*Y*) |
| 5 | *u-lock*(*Y*) |  | *s-lock*(*X*), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 6 | *read\_item(Y)* |  | *s-lock*(*X*), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 7 | *x-lock*(*Y*)  denied |  | *s-lock*(*X*), *u-lock*(*Y*) | *s-lock*(*Y*) |
| 8 | *unlock(X)* |  | *u-lock*(*Y*) | *s-lock*(*Y*) |
| 9 |  | *s-lock*(*X*) | *u-lock*(*Y*) | *s-lock*(*X*), *s-lock*(*Y*) |
| 10 |  | *read\_item(X)* | *u-lock*(*Y*) | *s-lock*(*X*), *s-lock*(*Y*) |
| 11 |  | *unlock(X)* | *u-lock*(*Y*) | *s-lock*(*Y*) |
| 12 |  | *unlock(Y)* | *u-lock*(*Y*) |  |
| 13 | *x-lock*(*Y*) |  | *u-lock*(*Y*), *x-lock*(*Y*) |  |
| 14 | *write\_item(Y)* |  | *u-lock*(*Y*), *x-lock*(*Y*) |  |
| 15 | *unlock(Y)* |  |  |  |

## Question 5 (10 marks)

For each of the following schedules, determine if the schedule is allowed by 2PL or not and explain your reasons for each schedule.

**Note.** Each schedule spans two lines of text.

**(a)** *S*1: *l*1(*X* ); *r*1(*X* ); *w*1(*X* ); *l*1(*Y* ); *u*1(*X* ); *l*2(*X* ); *r*2(*X* ); *r*1(*X* ); *w*1(*Y* ); *u*1(*Y* ); *l*2(*Y* ); *r*2(*Y* );

*w*2(*X* ); *w*2(*Y* ); *u*2(*X* ); *u*2(*Y* ) (2 marks)

Ans:

*S*1 is allowed by 2PL because in each transaction of *S*1, all lock operations precede all unlocks.

**(b)** *S*2: *l*1(*X* ); *l*1(*Y* ); *r*1(*X* ); *w*1(*X* ); *u*1(*X* ); *l*2(*X* ); *r*2(*X* ); *r*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *l*2(*Y* ); *r*2(*Y* );

*w*2(*X* ); *u*2(*X* ); *w*2(*Y* ); *u*2(*Y* ) (2 marks)

Ans:

*S*2 is allowed by 2PL because in each transaction of *S*1, all lock operations precede all unlocks.

**(c)** *S*3: *l*1(*X* ); *r*1(*X* ); *w*1(*X* ); *u*1(*X* ); *l*2(*X* ); *r*2(*X* ); *l*1(*Y* ); *r*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *l*2(*Y* ); *r*2(*Y* );

*w*2(*X* ); *w*2(*Y* ); *u*2(*X* ); *u*2(*Y* ) (2 marks)

Ans:

*S*3 is not allowed by 2PL because in transaction T1, *l*1(*Y* ) executes after *u*1(*X* ).

**(d)** *S*4: *l*1(*X* ); *r*1(*X* ); *w*1(*X* ); *u*1(*X* ); *l*1(*Y* ); *r*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *l*2(*X* ); *r*2(*X* ); *w*2(*X* ); *u*2(*X* );

*l*2(*Y* ); *r*2(*Y* ); *w*2(*Y* ); *u*2(*Y* ) (2 marks)

Ans:

*S*4 is not allowed by 2PL because in transaction T1, *l*1(*Y* ) executes after *u*1(*X* ).

**(e)** *S*5: *l*1(*X* ); *r*1(*X* ); *w*1(*X* ); *l*1(*Y* ); *u*1(*X* ); *r*1(*Y* ); *w*1(*Y* ); *u*1(*Y* ); *l*2(*X* ); *r*2(*X* ); *w*2(*X* ); *l*2(*Y* );

*u*2(*X* ); *r*2(*Y* ); *w*2(*Y* ); *u*2(*Y* ) (2 marks)

Ans:

*S*5 is allowed by 2PL because in each transaction of *S*1, all lock operations precede all unlocks.

## Question 6 (6 marks)

Examine the schedule given below. There are three transactions, *T*1, *T*2 and *T*3. Again, assume that the transactions use shared buffers.

Initially, the value of *X* = 1 and *Y* = 2. The assignments happen within the local memory space of the transactions and the effects of these assignments are not reflected in the database until the commit operation.

Assume that the undo/redo algorithm with simple checkpoints is used and that the log records up to now are on disk.

Determine what recovery is needed for each of the transactions *T*1, *T*2 and *T*3 if the system crashes with immediate effect at time *t* = 13 (at the end of line 13 but before the start of line 14).

(2 marks per transaction)

|  |  |  |  |
| --- | --- | --- | --- |
| **Time (*t*)** | **Transaction *T*1** | **Transaction *T*2** | **Transaction *T*3** |
| 0 |  |  | Start\_transaction |
| 1 |  |  | read\_item (*Y* ) |
| 2 |  |  | *Y* := *Y* + 1 |
| 3 | Start\_transaction |  |  |
| 4 | read\_item (*X* ) |  |  |
| 5 | *X* := *X* + 1 |  |  |
| 6 |  |  | write\_item (*Y* ) |
| 7 |  |  | commit |
| 8 |  | Start\_transaction |  |
| 9 |  | read\_item (*Y* ) |  |
| 10 |  | read\_item (*X* ) |  |
| 11 |  | *Y* := *Y* + *X* |  |
| 12 |  | write\_item (*Y* ) |  |
| 13 |  | commit |  |
| 14 | read\_item (*Y* ) |  |  |
| 15 | *Y* := *Y* + *X* |  |  |
| 16 | write\_item (*X* ) |  |  |
| 17 | checkpoint | | |
| 18 | commit |  |  |

Ans:

|  |
| --- |
| Log record on the disk: |
| <START T3> |
| <START T1> |
| <T3, Y, 2, 3> |
| <COMMIT T3> |
| <START T2> |
| <T2, Y, 3, 4> |
| <COMMIT T2> |

The recovery manager processes the log in reverse order:

|  |  |  |
| --- | --- | --- |
| Step | Current log record | Recovery manager’s action |
| 1 | <COMMIT T2> | Remember for later steps that T2 has finished. |
| 2 | <T2, Y, 3, 4> | Do nothing, because T2 has finished. |
| 3 | <START T2> | Do nothing. |
| 4 | <COMMIT T3> | Remember for later steps that T3 has finished. |
| 5 | <T3, Y, 2, 3> | Do nothing, because T3 has finished. |
| 6 | <START T1> | Undo T1. |
| 7 | <START T3> | Do nothing. |

|  |
| --- |
| The final log on disk looks like this: |
| <START T3> |
| <START T1> |
| <T3, Y, 2, 3> |
| <COMMIT T3> |
| <START T2> |
| <T2, Y, 3, 4> |
| <COMMIT T2> |
| <ABORT T1> |

Conclusion:

T1: There is need to apply undo/redo recovery.

T2: There is no need to apply undo/redo recovery.

T3: There is no need to apply undo/redo recovery.