Fachbereich für Computerwissenschaften

Prof. Dr. Nikolaus Augsten
Jakob-Haringer-Str. 2
5020 Salzburg, Austria
Telefon: +43 662 8044 6347
E-Mail: nikolaus.augsten@sbg.ac.at

PARIS LODRON UNIVERSITÄT SALZBURG

Similarity Search in Large Databases — Prüfung
Wintersemester 2022/2023 — Mock Exam

Name: _____ **Matrikelnummer:** _____

Hinweise

- Bitte überprüfen Sie die Vollständigkeit des Prüfungsbogens (29 nummerierte Seiten).
- Schreiben Sie Ihren Namen und Ihre Matrikelnummer auf jedes Blatt des Prüfungsbogens und geben Sie alle Blätter ab.
- Grundsätzlich sollten Sie alle Antworten auf den Prüfungsbogen schreiben.
- Sollten Sie mehr Platz für eine Antwort benötigen, bitte einen klaren Verweis neben die Frage auf die Seitennummer des zusätzlichen Blattes setzen.
- Keinen Bleistift verwenden. Keinen roten Stift verwenden.
- Verwenden Sie die Notation und die Lösungsansätze, die während der Vorlesung besprochen wurden.
- Aufgaben mit mehr als einer Lösung werden nicht bewertet.
- Als Unterlage ist ein beliebig (auch beidseitig) beschriftetes A4-Blatt erlaubt.
- Zeit für die Prüfung: 90 Minuten

Unterschrift _____

Korrekturabschnitt — Bitte frei lassen

Aufgabe	1	2	3	4	5	6	7	8	Summe
Maximale Punkte	2	2	2	2	2	2	2	2	16
Erreichte Punkte									

Exercise 1 — 1 Point

Can the following schedule be the output of a **strict two-phase locking** scheduler? If yes, add all required lock/unlock instructions. Otherwise, explain why.

```
T1:         T2:         T3:
_____________________________

read(X)

read(Y)

            read(X)

                        read(X)

                        write(X)

write(X)

            write(Y)

            COMMIT

                        read(X)

read(X)

                        COMMIT

COMMIT
_____________________________
```

Exercise 2 — 1 Point

Consider the following schedule and the **two-phase locking** scheduler.

T1:	T2:	T3:
read(X)		
	write(X)	
		read(X)
write(X)		
		write(X)

Does the schedule result in a deadlock?

Exercise 3 — 1 Point

Can the following schedule be the output of a strict two-phase locking scheduler? If yes, show the schedule with all required lock and unlock instructions. Otherwise, explain why.

```
T1:         T2:         T3:
_____________________________

read(X)

read(Y)

                        read(X)

                        write(X)

            read(X)

read(Z)

write(Z)

COMMIT

            write(Y)

            COMMIT

                        read(V)

                        COMMIT
_____________________________
```

Exercise 4 — 1 Point

Does the following schedule adhere to the **two phase locking (2PL)** protocol? If **yes**, show the schedule with all required lock and unlock instructions. If **not**, explain why 2PL is violated.

```
T1:          T2:          T3:
_____________________________

                          read(C)

read(A)

write(B)

             read(B)

                          write(A)

_____________________________
```

Exercise 5 — 1 Point

Can the following schedule be the output of a **two-phase** locking scheduler? If so, show the schedule with all required lock and unlock instructions. Otherwise explain why. Could it be the output of a ***strict* two-phase** locking scheduler? Why (not)?

T1:	T2:	T3:
read(A)		
write(A)		
	read(B)	
	write(C)	
	read(A)	
	COMMIT	
		read(C)
		write(C)
		COMMIT
read(B)		
COMMIT		

Exercise 6 — 1 Point

Can the following schedule be the output of a **strict two-phase locking** scheduler? If yes, add all required lock/unlock instructions. Otherwise, explain why.

```
T1:        T2:        T3:
_____________________________

           read(A)

           read(B)


                      read(A)

                      write(A)

           write(B)

           COMMIT


                      read(A)

                      read(B)

read(B)

read(A)

                      COMMIT

COMMIT
_____________________________
```

Exercise 7 — 1 Point

Is the following schedule **conflict serializable**? If it is, give an equivalent serial schedule. If it is not, explain why.

T1:	T2:	T3:	T4:
			read(B)
	read(A)		
	write(A)		
read(C)			
			read(A)
		read(C)	
		read(B)	
			read(C)
		write(A)	
		write(B)	
write(B)			

Exercise 8 — 1 Point

Consider the following schedule.

(a) Which transaction has to abort to trigger a **cascading rollback**? Why?

(b) Insert `COMMIT` instructions such that the schedule becomes cascadeless.

T1:	T2:	T3:	T4:
	read(A)		
	write(A)		
write(B)			
read(A)			
			read(B)
		read(C)	
			write(C)
		read(C)	

Exercise 9 — 1 Point

Consider the following schedule.

(a) Which transaction has to abort to trigger **cascading rollback**?

(b) Insert `COMMIT` instructions such that the schedule becomes cascadeless.

T1:	T2:	T3:
	`read(A)`	
	`write(A)`	
`write(B)`		
`read(A)`		
		`read(A)`
		`read(B)`

Exercise 10 — 1 Point

Identify which of the properties: **conflict serializable**, **recoverable**, **cascadeless**, are fulfilled by the following schedule. If a property is not fulfilled, explain why.

T1:	T2:	T3:	T4:
read(A)			
write(A)			
		read(C)	
	write(B)		
	write(C)		
		read(A)	
			read(B)
read(A)			
	COMMIT		
		write(A)	
			write(B)
			COMMIT
read(B)			
COMMIT			
		COMMIT	

Exercise 11 — 1 Point

Consider the following schedule. State if it is **conflict serializable** and explain why. Give an equivalent serial schedule if possible.

T1:	T2:	T3:	T4:
read(X)			
	write(X)		
		read(X)	
read(Y)			
	write(Y)		
read(Z)			
		write(Z)	
			read(Z)
			write(Z)

Exercise 12 — 1 Point

Is the following schedule **conflict serializable**? Draw the precedence graph to verify. If it **is not**, explain why. If it **is**, give an equivalent serial schedule.

T1:	T2:	T3:	T4:
	write(C)		
			read(B)
		read(C)	
		write(C)	
			read(A)
read(C)			
			write(A)
	read(A)		
			write(B)
read(B)			
		write(A)	

Exercise 13 — 1 Point

Is the following schedule **conflict serializable**? Draw a precedence graph to verify.

```
T1:         T2:         T3:
            write(C)
            read(A)
                        write(C)
read(B)
            write(A)
                        read(B)
            read(A)
write(B)
read(C)
                        write(B)
                        read(A)
```

Exercise 14 — 1 Point

Is the following schedule **conflict serializable**? Draw a precedence graph to verify.

```
T1:       T2:       T3:
                    read(B)
                    write(A)
read(A)
          write(A)
read(B)
write(A)
COMMIT
          read(A)
          COMMIT
                    read(B)
                    COMMIT
```

Exercise 15 — 1 Point

Is the following schedule **conflict serializable**? Draw a precedence graph to verify. If it is not, explain why. If it is, give an equivalent serial schedule.

T1:	T2:	T3:	T4:
		read(A)	
read(A)			
	write(C)		
read(B)			
write(A)			
			read(C)
			write(B)
	read(A)		
	write(A)		
		read(B)	

Exercise 16 — 1 Point

Is the following schedule **conflict serializable**? Draw the precedence graph to verify. If it is not, explain why. If it is, give an equivalent serial schedule.

T1:	T2:	T3:	T4:
	write(C)		
			read(B)
			read(D)
		read(C)	
		write(C)	
			read(A)
read(A)			
	write(A)		
			write(B)
		write(B)	
write(D)			
		write(A)	

Exercise 17 — 1 Point

Consider the following schedule and the **two-phase locking** scheduler.

(a) Assume that all transactions want to issue a `write` operation on item `A` in the following order: `T1`, `T2`, `T3`. Complete the schedule by inserting lock requests (e.g., `R:lock-S(A)`) and granted locks (e.g., `G:lock-S(A)`) for all remaining `read` and `write` operations. Denote in the schedule if a transaction has to wait since another transaction holds an incompatible lock.

(b) Draw the wait-for graph. Does the schedule result in a deadlock? Why?

```
T1:          T2:          T3:
_____________________________________
R:lock-S(A)
G:lock-S(A)
read(A)
_____________________________________

                          read(A)
_____________________________________

_____________________________________

_____________________________________

_____________________________________
```

Exercise 18 — 1 Point

Is the following schedule in a **deadlock state**? Draw the wait-for graph to verify. If it **is** in a a deadlock state, propose a way to recover from the deadlock.

T1:	T2:	T3:
lock-S(A)		
read(A)		
		lock-X(B)
		lock-S(A)
	lock-X(C)	
		read(A)
lock-X(B)		
	read(C)	
	lock-X(A)	
		lock-S(C)

Exercise 19 — 1 Point

Consider the following schedule and the **two-phase locking** scheduler.

(a) Assume that all transactions want to issue a `write` operation on item `A` in the following order: `T3`, `T2`, `T1`. Complete the schedule by inserting lock requests (e.g., `R:lock-S(A)`) and granted locks (e.g., `G:lock-S(A)`) for all remaining `read` and `write` operations. Denote in the schedule if a transaction has to wait since another transaction holds an incompatible lock.

(b) Draw the wait-for graph. Does the schedule result in a deadlock? Why?

```
T1:          T2:          T3:
_____________________________________
R:lock-S(A)
G:lock-S(A)
read(A)
_____________________________________


             read(A)
_____________________________________

_____________________________________

_____________________________________

_____________________________________
```

Exercise 20 — 1 Point

Consider the following schedule.

(a) Which transaction has to abort to trigger a **cascading rollback**?

(b) Insert `COMMIT` instructions such that the schedule becomes cascadeless.

(c) Why is cascadelessness of a schedule desirable?

T1:	T2:	T3:
		`read(A)`
		`write(B)`
		`write(A)`
`read(A)`		
`write(B)`		
	`read(B)`	

Exercise 21 — 1 Point

Consider the following schedule.

(a) Which transaction must abort to trigger a **cascading rollback** of all other transactions?

(b) Show the commit order (by inserting the commit commands into the schedule) such that the schedule is **cascadeless**.

T1:	T2:	T3:	T4:
	read(B)		
read(D)			
	write(A)		
			read(A)
			write(C)
		read(E)	
read(C)			
write(B)			
		read(B)	

Exercise 22 — 1 Point

Consider the following schedule.

(a) Which transaction has to abort to trigger **cascading rollback**?

(b) Insert `COMMIT` instructions such that the schedule becomes cascadeless.

T1:	T2:	T3:
	read(A)	
	write(A)	
write(B)		
read(A)		
		read(A)
		read(B)

Exercise 23 — 1 Point

Answer the following questions (be concise):

1. Explain three *undesirable phenomena* of concurrent transactions.
2. What does *read uncommitted* stand for?
3. What is a *lost update* and how can it be prevented?
4. What does Read Committed stand for?
5. What does **Repeatible Read** stand for?
6. What is a **Deadlock** (give and example) and how can it be detected?

Exercise 24 — 1 Point

Consider the following schedule. Add 4 `COMMIT` instructions such that the resulting schedule is **not recoverable**. Can it still be cascadeless? Why (not)? Also, state if it is **conflict serializable** and give an equivalent serial schedule if possible.

T1:	T2:	T3:	T4:
	write(X)		
		read(X)	
read(X)			
	write(Y)		
read(Z)			
			read(Y)
			write(Z)

Exercise 25 — 1 Point

Identify which of the properties: **conflict serializable**, **recoverable**, **cascadeless**, are fulfilled by the following schedule. If a property is not fulfilled, explain why.

T1:	T2:	T3:	T4:
read(A)			
			write(B)
		read(C)	
	read(B)		
	write(A)		
			COMMIT
	COMMIT		
		read(A)	
read(B)			
write(B)			
		write(A)	
		read(B)	
COMMIT			
		COMMIT	

Exercise 26 — 1 Point

Identify which of the properties: **serial**, **conflict serializable**, **recoverable**, **cascadeless**, are fulfilled by the following schedule. If a property is not fulfilled, explain why.

T1:	T2:	T3:	T4:
			write(B)
		read(C)	
read(A)			
			COMMIT
	read(B)		
	write(A)		
	COMMIT		
		read(A)	
read(B)			
		write(A)	
write(B)			
COMMIT			
		read(B)	
		COMMIT	

Exercise 27 — 1 Point

Show the commit order (by inserting the commit commands into the schedule) for the following schedule such that the schedule is **cascadeless**.

T1:	T2:	T3:	T4:
read(B)			
	write(A)		
			read(A)
			write(A)
		read(A)	
write(B)			
		read(B)	

Exercise 28 1 Point

Show the commit order (by inserting the commit commands into the schedule) for the following schedule such that the schedule is **recoverable**.

T1:	T2:	T3:	T4:
read(B)			
		write(A)	
			read(A)
	read(A)		
write(B)			
			write(A)
			read(B)
	read(A)		