Department of Computer Science

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

PARIS LODRON UNIVERSITÄT SALZBURG

Advanced Databases — Exam
Winter Semester 2025/26 — 25.02.2026

Name: _____ **Student ID:** _____

Hints

- Check whether you received all pages of the exam (9 pages).
- Write your name or your student ID on each sheet of the exam and hand in all pages.
- All answers are expected to be written on the exam sheets.
- Clearly highlight and enumerate additional pages that are used for longer answers. Match your text with the according exercise.
- Only use pencils that are permanent and non-red colored.
- Use the notation and techniques discussed in the lecture.
- Exercises with more than one solution are not graded.
- **Multiple-Choice**: Wrong answers result in point deductions!
- You are allowed to use one A4 sheet with your personal notes (both sides, hand written or printed).
- Exam duration: 90 minutes

Signature _____

Grading — Filled by the examiner

Exercise	1	2	3	4	5	6	7	8	Sum
Total points	1	1	1	1	1	1	1	1	8
Points reached									

Exercise 1 — 1 Point

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

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

Exercise 2 — 1 Point

Consider the following schedule and a *strict* two-phase locking scheduler *with* lock conversions. Insert lock, unlock, upgrade, and commit instructions such that all read and write operations in the schedule can be performed in this order. **Avoid unlock and commit operations that are not necessary.**

```
T1:        T2:        T3:        T4:
_____________________________________

           read(A)
           write(A)



                                  read(B)

read(B)

                      read(A)



                                  write(B)


read(A)
write(A)

_____________________________________
```

Exercise 3 — 1 Point

Consider the following schedule. Indicate what happens when the schedule is processed by a **multiversion timestamp-ordering** scheduler. The transactions start in order with TS(T1)=1, TS(T2)=2, TS(T3)=3, TS(T4)=4. Assume that initially no versions of data item A exist. State if a transaction must be rolled back and also consider cascading rollbacks.

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

Exercise 4 — 1 Point

Consider the following **multiversion two-phase locking** schedule. Assume that the starting value of `ts-counter` and the initial versions of data items A and B are 0. T_i and T_j are read-only transactions and start with their first read operation.

For each row in the schedule, indicate any

- changes to locks,
- waiting transactions,
- newly created data items,
- changes to versions of data items,
- and the data item version for reads.

Also, provide

- the timestamps of the read-only transactions
- and the final value of `ts-counter`.

Ti:	Tj:	Tk:	Tl:
		write(B)	
	read(B)		
		write(A)	
			write(B)
		commit	
read(A)			
			write(B)
read(B)			
			commit
read(B)			
	read(A)		

Exercise 5 — 1 Point

Consider the following schedule and the **validation based** scheduler.

```
T1:       T2:       T3:       T4:
──────────────────────────────────────
start
──────────────────────────────────────
          start
──────────────────────────────────────
                              start
──────────────────────────────────────
                    start
──────────────────────────────────────
          validate
──────────────────────────────────────
          finish
──────────────────────────────────────
validate
──────────────────────────────────────
finish
──────────────────────────────────────
                    validate
──────────────────────────────────────
                    finish
──────────────────────────────────────
                              validate
──────────────────────────────────────
                              finish
──────────────────────────────────────
```

The objects in the database that can be read or written are: A, B, C, D, E, F.

The read and write sets of the transactions are:

T1: R-set(T1)={A,B}, W-set(T1)={C,D}
T2: R-set(T2)={A,C}, W-set(T2)={D,F}
T3: R-set(T3)={C,E}, W-set(T3)={B,F}

Answer the following questions.

(1) Can `T2` be committed? Explain which conditions are satisfied or which one is violated.
(2) Suppose that the validation of T4 and all previous validations succeeded. What could have been the largest read set of T4?

Exercise 6 — 1 Point

Consider the following log file (physical logging). Reconstruct a **possible schedule** that could have resulted in this log file, including **starts**, **commits**, and **rollbacks** of transactions, possible system **crashes**, and **checkpoints**. What are the **initial** and **final** values of A, B, and C?

```
<T1, start>
<T1, A, 100, 150>
<T2, start>
<T2, B, 100, 30>
<T3, start>
<T1, commit>
<checkpoint, {T2, T3}>
<T3, B, 30, 80>
<T3, B, 30>
<T3, abort>
<T2, B, 100>
<T2, abort>
<T4, start>
<T4, C, 200, 150>
<T4, commit>
```

Exercise 7 — 1 Point

Why does the following schedule **NOT** adhere to the **timestamp-ordering** protocol?

TS(T1)=1, TS(T2)=2, TS(T3)=3

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

Exercise 8 — 1 Point

Consider the following schedule and **physical logging** with **log buffering**. The log buffer can hold 4 log entries. Initially, A=100 and B=200.

(1) After each creation of a log record, show the **content** of the log buffer and state whether the buffer has to be **flushed**. **Indicate** when each transaction may safely enter the "committed" state. Avoid as many disk writes as safely possible.

(2) Assume the case that the system **crashes** directly before the commit of T_2. Which log entries are **guaranteed** to be on stable storage at crash point?

```
T1:       T2:
start
R(A)
A:=A+50
W(A)
          start
          R(B)
          B:=B+50
          W(B)
commit
          R(A)
          A:=A+50
          W(A)
          commit
```