Exercise 9 (Serializability) Consider the following transactions:

- T1: read(A,t); t:=t+2; write(A,t); read(B,t); t:=t*3; write(B,t)
- T2: read(B,s); s:=s*2; write(B,s); read(A,s); s:=s+3; write(A,s)

As usual, we assume that whatever consistency constraints are given, these are preserved when T1 and T2 execute in isolation.

a) Show that for the above T1, T2 the schedules (T1;T2) and (T2;T1) have the same effect on the database (this is usually *not* the case).

> *result*$(T1;T2):\ A_1 := A_0 + 2; B_1 := 3 * B_0; B_2 := 2 * B_1; A_2 := A_1 + 3$
> $\Rightarrow A_2 = A_0 + 5; B_2 = 6 * B_0$; *the same result state* (A_2, B_2) *is reached for* $(T2;T1)$.

b) Give a serializable and a non-serializable schedule for T1 and T2.

> *T1 modifies A, then B; T2 the other way round. The serial schedules above are trivially serializable. A really interleaved serializable schedule is* $S = read_1(A,t); read_2(B,s); t_1 := t_1+2; s_2 := s_2 * 2; write_2(B,s); write_1(A,t); read_2(A,s); read_1(B,t); t_1 := t_1 * 3; write_1(B,t); s_2 := s_2 + 3; write_2(A,s)$. *But note that S is not conflict serializable!*
> *A non-serializable schedule is obtained, e.g., by moving T2's update on A between the read and write on A of T1. Then T2's effect on A is lost!*

c) How many serial schedules are there?

> *A schedule is* ***serial*** *if for any two actions* p_i *from* T_i *and* p_j *from* T_j *follows: if* $p_i <_S p_j$ *then all* T_1 *actions are before all* T_j *actions. Otherwise it is* ***interleaved****. For two transactions, there are two serial schedules* $(T1;T2)$ *and* $(T2;T1)$.

d) How many serializable schedules are there?

> *NB: Here we do* ***not*** *ask for conflict serializable schedules, but just serializable schedules (i.e., which will always produce the same result as some serial schedule). In this example, S is* ***not*** *serializable, if a transaction T' reads the old value (A or B) after T has read but before T has written the new value. Then one update will be lost. E.g., we cannot move the r(A) of T2 before the w(A) but after the read of T1.*
> *However the first 3 actions of T1 and T2 can be freely interleaved, similary, the last 3 actions. So we get* $\binom{6}{3}\binom{6}{3} + 2 = 402$ *serializable schedules (including the two serial schedules); cf. Exercise 11.*

Exercise 10 (Serializability) Consider the schedule $S =$

$$r1(A); r2(A); r3(B); w1(A); r2(C); r2(B); w2(B); w1(C)$$

a) Give the precedence graph $P(S)$.

> $T3 \rightarrow T2 \rightarrow T1$

b) Is the schedule conflict-serializable? If so, give all equivalent serial schedules.

> *Yes, since P(S) is acyclic. The only equivalent serial schedule is* $(T3;T2;T1)$.

Exercise 11 (Interleavings) Given two transaction $T1$ and $T2$ consisting of $n1$ and $n2$ actions. How many interleavings (i.e., different schedules with $T1$ and $T2$) are possible?

> *Any schedule for T1 and T2 has n1+n2 "positions"; n1 of those will be used for T1 actions: we can choose those n1 arbitrarily from all positions; after that everything is fixed. Hence there will be* $\binom{n1+n2}{n1} = \binom{n1+n2}{n2} = \frac{(n1+n2)!}{n1!n2!}$ *of those.*

Exercise 12 (Serializability, s.154) Show that the converse of the Lemma does not hold, i.e., $P(S1) = P(S2)$ does *not* imply that $S1$ and $S2$ are conflict equivalent.

Consider $S1 = w1(A); r2(A); w2(B); r1(B)$ and $S2 = r2(A); w1(A); r1(B); w2(B)$. Then $P(S1) = P(S2) = \{T1 \rightarrow T2 \rightarrow T1\}$ but we cannot swap non-conflicting actions to obtain one schedule from the other.

Exercise 13 (Deadlocks, Starvation, s.163ff) We speak of *starvation* if a transaction is repeatedly rolled back and never finishes. Can the following anti-deadlock strategies lead to starvation (explain/give an example):

- transaction which have reached their time limit are rolled back

  Can lead to starvation: e.g., if the limit is too short even for a non-deadlocked execution.

- transactions which would introduce a cycle in the waits-for graph are rolled back

  Can lead to starvation: after roll back, a transaction may again become the one that introduces a cyclic wait.

- lock request can appear only in a predefined order

  Cyclic wait conditions are prevented in the first place, so transactions are never rolled back. Can they wait infinitely? Not if we have a "fair" scheduler, which allows transactions to make progress eventually. For example, if we distribute available locks on a first-come first-served basis, starvation is prevented.

- transactions are rolled back based on their timestamps (WAIT-DIE or WOUND-WAIT scheme)

  Starvation is prevented since eventually any transaction will become oldest and thus cannot be rolled back.

Exercise 14 (Deadlocks, s.167) Show that the WAIT-DIE and WOUND-WAIT strategies prevent deadlocks.

A deadlock is a cycle in the waits-for graph. However, for each of the schemes, edges only go in one direction (either wait for older or wait for younger). Hence there cannot be cycles.

Exercise 15 (Serializability) Consider the following two transactions:

- T_0: r(A); r(B); if A=0 then B:=B+1; w(B);
- T_1: r(B); r(A); if B=0 then A:=A+1; w(A);

Assume initial values A=B=0, and that we have the consistency requirement: A=0 OR B=0.

a) Show that every serial execution maintains consistency.

b) Give a nonserializable schedule.

c) Is there a serialiable, non-serial schedule?

Exercise 16 (2PL, s.162) Show that the schedule in s.162 is

a) serializable and

b) explain why it cannot be obtained from a 2PL scheduler.

Exercise 17 (View/Conflict Serializability) Consider the following schedule S:

T_1	T_2	T_3
$w(A)$		
	$r(A)$	
		$w(B)$
$w(B)$		
		$w(B)$
	$w(A)$	
		$r(B)$
	$r(B)$	

a) Give the precedence graph $P(S)$.

b) Is S conflict serializable? Explain.

c) Is S view serializable? Explain.

$P(S) = T_1 \rightarrow T_2, T_1 \rightarrow T_3, T_3 \rightarrow T_1$, i.e., cyclic, so not conflict serializable.

For view serializability, add T_b and T_f:

T_b	T_1	T_2	T_3	T_f
$w(A)$				
$w(B)$				
	$w(A)$			
		$r(A)$		
			$w(B)$	
	$w(B)$			
			$w(B)$	
		$w(A)$		
			$r(B)$	
		$r(B)$		
				$r(A)$
				$r(B)$

and construct the labeled precedence graph. The crucial edges are $T_1 \xrightarrow{B,1} T_3$ and $T_2 \xrightarrow{B,1} T_1$. By picking the former instead of the latter, we get an acyclic graph; its topological sort yields the serial schedule $(T_1; T_3; T_2)$

Exercise 18 (Serializability) Two transactions are *not interleaved* in a schedule S if every operation of one transaction precedes every operation of the other in S. (Note: S need not be serial.) Give an example of a *serializable* schedule S with the following properties:

- T_1 and T_2 are not interleaved in S, and
- T_1 precedes T_2 in S, and
- in any serial schedule equivalent to S, T_2 precedes T_1.

Hint: S may have additional transactions.

Consider this schedule:

T_1	T_2	T_3
		$w(A)$
$r(A)$		
	$r(B)$	
		$w(B)$

In an equivalent serial schedule we have to make sure that $T_3 < T_1$ (because of $w_2(A), r_1(A)$) and that $T_2 < T_3$ (because of $r_2(B), w_3(B)$). Hence we have $(T_2; T_3; T_1)$.

Exercise 19 (IO Cost, s.115ff) Consider two relations R and S with T(R) = 1000, T(S) = 500, and S(R) = S(S) such that 10 tuples fit into a block. Determine the number of disk IOs for the following joins:

a) nested-loop join with (i) non-clustered, tuple access, (ii) non-clustered, block access, (iii) clustered, block access,

b) merge join with clustered relations (i) given R, S sorted, (ii) R, S not sorted,

c) index join: consider the different expected number of matching R tuples from s.129, i.e., 1, 2, and 0.01

d) hash join with ideal distribution across buckets.

Exercise 20 (Join Algorithms) Which are typical "good uses" of nested-loop join, hash join, merge join, and index join algorithms?

nested loop: small relations only; hash: equi-joins on unsorted, non-indexed relations; merge: sorted relations and non-equi joins like $R.A > S.B$; index: if index available, but consider expected number of matches

Exercise 21 (Recovery)

a) Undo logging requires that before an item is modified on disk, the log records pertaining to X are on disk (WAL: write ahead logging). Show using an example that an inconsistent database may result if log records for X are not flushed to disk before X.

b) Show using an example that an inconsistent database may result if some items are not written to disk before the commit is written to the log (even if WAL holds).

c) Redo logging: show using an example that an inconsistent database may result if some items are written to disk before the commit is written on the log (even if WAL holds)