Question 1: Schedules - RC, ACA, Strict (1 P.)

For the following schedules, decide in which of the classes RC, ACA, or ST they are.

$$\begin{aligned} s_1 &:= w_1(b)\ w_1(a)\ r_1(b)\ w_2(a)\ w_2(b)\ c_1\ r_2(a)\ w_2(a)\ c_2 \\ s_2 &:= r_1(b)\ w_1(d)\ w_1(c)\ r_2(c)\ r_3(a)\ r_3(c)\ w_2(a)\ w_2(d)\ c_1\ r_2(b)\ c_2\ r_3(b)\ c_3 \\ s_3 &:= r_3(a)\ w_3(b)\ r_3(c)\ c_3\ w_2(b)\ w_2(c)\ r_1(d)\ r_1(c)\ w_1(c)\ w_2(a)\ c_1\ c_2 \end{aligned}$$

Required submission: Class for each schedule; Reasoning for class;

Question 2: 2-Phase-Locking and Waits-for-Graph (1 P.)

a) Given the following schedules:

$$\begin{aligned} s_1 &= w_4(a)\ r_2(c)\ r_3(a)\ r_3(c)\ c_3\ r_2(a)\ r_4(c)\ c_4\ w_2(a)\ c_2 \\ s_2 &= w_1(b)\ r_2(c)\ w_3(a)\ w_2(b)\ r_1(a)\ r_3(b)\ c_1\ c_3\ w_2(a)\ c_2 \end{aligned}$$

Create a 2PL history for both schedules and note the Waits-for-Graph (WfG) every time it changes. If you encounter a deadlock, abort the transaction that caused the deadlock (No deadlock prevention strategy).

Required submission: 2PL histories; WfGs after every change;

b) Specify a 2PL history and WfG (up until a deadlock) involving four transactions t_1, t_2, t_3, and t_4, such that the strategies **most cycles** and **most edges** cause the same transaction to be aborted as the strategy **youngest**. Assume that the strategies are applied as soon as the deadlock is detected.

Required submission: 2PL history; WfG

Question 3: Deadlock prevention (1 P.)

Implement a scheduler, which creates a (SS)2PL schedule, using the deadlock prevention strategies wait-die, wound-wait and immediate restart.

Test your program with the given schedule. It has to print the output schedule for each of the prevention strategies and indicate when a transaction has to wait.

$$s := w_1(x)\ r_2(x)\ w_3(y)\ r_1(y)\ r_3(z)\ w_1(x)\ c_1\ w_2(y)\ c_2\ w_3(y)\ c_3$$

You can use the template provided in OLAT, which implements an internal representation of a history and all required operations.

Required submission: Source code; Output after executing the code;

Question 4: Intention Locks (1 P.)

Additionally to read and write locks we now consider "intention locks". With the following query, an application could indicate that the returned tuples will soon be updated:

```
SELECT * FROM parts WHERE p_partkey = 1234 FOR UPDATE;
```

a) Describe which additional rules could be implemented given these new locks. Give a possible compatibility matrix, including the known and the new locks.

   Notation read with update intention: $\tilde{r}_i(x)$; Intention lock: $il_i(x)$.

   Required submission: Rule(s); Compatibility matrix;

b) Give a possible output history for the following schedule:

$$s = r_3(y)\ r_1(x)\ \tilde{r}_3(x)\ r_1(y)\ \tilde{r}_2(y)\ r_4(x)\ r_4(z)\ a_4\ c_1\ w_3(x)\ w_2(y)\ c_2\ c_3$$

   Required submission: History

Question 5: Timestamp-Based Approaches (1 P.)

$$s_1 = w_1(x)\ w_2(x)\ r_2(x)\ r_1(x)$$
$$s_2 = r_1(y)\ r_2(y)\ r_1(x)\ w_2(y)\ w_3(x)\ w_1(x)\ r_1(x)$$

Describe how a timestamp ordering (TO) based scheduler would execute the operations. Complete the following tables. Note the used rule in the "Comment" column. The max-r and max-w columns contain the value of max-q-scheduled for the given object. One row contains the entry after applying the rule. You may assume $ts(t_1) < ts(t_2)$. Do not restart aborted transactions.

Required submission: Filled out tables

s_1:

Operation	max-r(x)	max-w(x)	Comment
BOT_1	$-\infty$	$-\infty$	$ts(t_1) = 1$
BOT_2	$-\infty$	$-\infty$	$ts(t_2) = 2$

s_2 <u>without</u> Thomas' write rule:

Operation	max-r(x)	max-w(x)	max-r(y)	max-w(y)	Comment
BOT_1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_1) = 1$
BOT_2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_2) = 2$
BOT_3	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_3) = 3$

s_2 <u>with</u> Thomas' write rule:

Operation	max-r(x)	max-w(x)	max-r(y)	max-w(y)	Comment
BOT_1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_1) = 1$
BOT_2	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_2) = 2$
BOT_3	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$ts(t_3) = 3$