1a) SELECT *
FROM (SELECT * FROM Countries WHERE CountryID >= 180
INNER JOIN Persons USING (CountryID)
INNER JOIN Devices USING (PersonID))

In relational algebra this is equivalent to:

$$\underbrace{\left[\sigma_{CountryID >= 180}(Countries)\right]}_{A} \bowtie \underbrace{Persons}_{B} \bowtie \underbrace{Devices}_{C}$$

$$n_{\sigma_{CountryID>=180}(Countries)} = n_{Countries} \frac{Max(CountryID) - 180 + 1}{Max(CountryID) - Min(CountryID) + 1} = \frac{200 - 180 + 1}{\cancel{200}} = 21$$

[redacted]

$$n_{A \bowtie B} = \min\left(\frac{n_A * n_{Persons}}{V(CountryID, A)}, \frac{n_A * n_{Persons}}{V(CountryID, Persons)}\right)$$

$$= \min\left(\frac{21 \times 100.000}{\cancel{21}}, \frac{21 \times 100.000}{200}\right) = 10500$$

[redacted]

$$n_{(A \bowtie B) \bowtie Devices} = \min\left(\frac{n_{A \bowtie B} * n_{Devices}}{V(PersonID, n_{A \bowtie B})}, \frac{n_{A \bowtie B} * n_{Devices}}{V(PersonID, Devices)}\right)$$

$$= \min\left(\frac{10.500 \times 200.000}{10.500}, \frac{10.500 \times 200.000}{100.000}\right)$$

$$= 10.500 \times 2 = 21.000$$

[redacted]

1b)

T1	T2
	1. LOCK-X(Countries[351]) UPDATE ... UNLOCK-X(Countries[351]) [redacted]
2. LOCK-X(Countries[351]) UPDATE ... UNLOCK-X(Countries[351]) [redacted]	
	3. LOCK-X(Countries[CountryName=='Portugallo']) DELETE.. 4. UNLOCK-X(...) [redacted]
	4.
5. LOCK-S(Countries) SELECT ... [redacted]	
6. UNLOCK-S(Countries) ABORT	

[redacted]

The scheduling does not obey to 2PL since locks are performed after the release of locks (e.g. in 1 and 2, [redacted]

The schedule is also not recoverable because in step 3 the record changed in step 2 is read before T1 has committed, in fact the problem is clear since T1 fails and thus the change in T2 has been rolled-back

[redacted]

1c)

	Log	Mem	Disk
1.	[redacted]		[redacted] R(123).end=NULL, R(124).end=NULL, Calls(726).cost=0.?, Calls(727).cost=0.
2.		R(123).end='2025-06-12 08:30' S	
3.			
4.	[redacted]	Calls(727).cost=1.75 S	
5.		Calls(726).cost=0.35 S	
6.		Calls(727).cost=2.05 S	R(123).end='2025- ... 08:30', C(726).cost=0.35, C(727).cost=2.0 [redacted]
7.	X={T1,T2}		
8.	[redacted]		
9.		R(124).end='2025-... 08:32' [redacted]	
10.			
11.	<T3, R(124).end, NULL>	R(124).end=NULL	
12.	[redacted]		
13.	CRASH		REDO PHASE
REDO	[redacted] REDO 9 UNDO-LIST={T2} REDO 11	R(124).end='2025-... 08:32' [redacted] R(124).end='NULL'	
	UNDO-LIST={T2}		UNDO PHASE
14.	[redacted] <T2, C(727).cost, 1.75>	C(727).cost=1.75 [redacted]	
15.	<T2, C(727).cost, 0.0>	C(727).cost=0.0	
16.	<T2, abort>		

1d)

1.

2.

3.

4. → N:=2

5.

6.

7.
7a → 2
8.
8a → 2
9.

[redacted]

~~Yes~~ the schedule is serializable [redacted] by executing T2 first followed by T1 or T3:

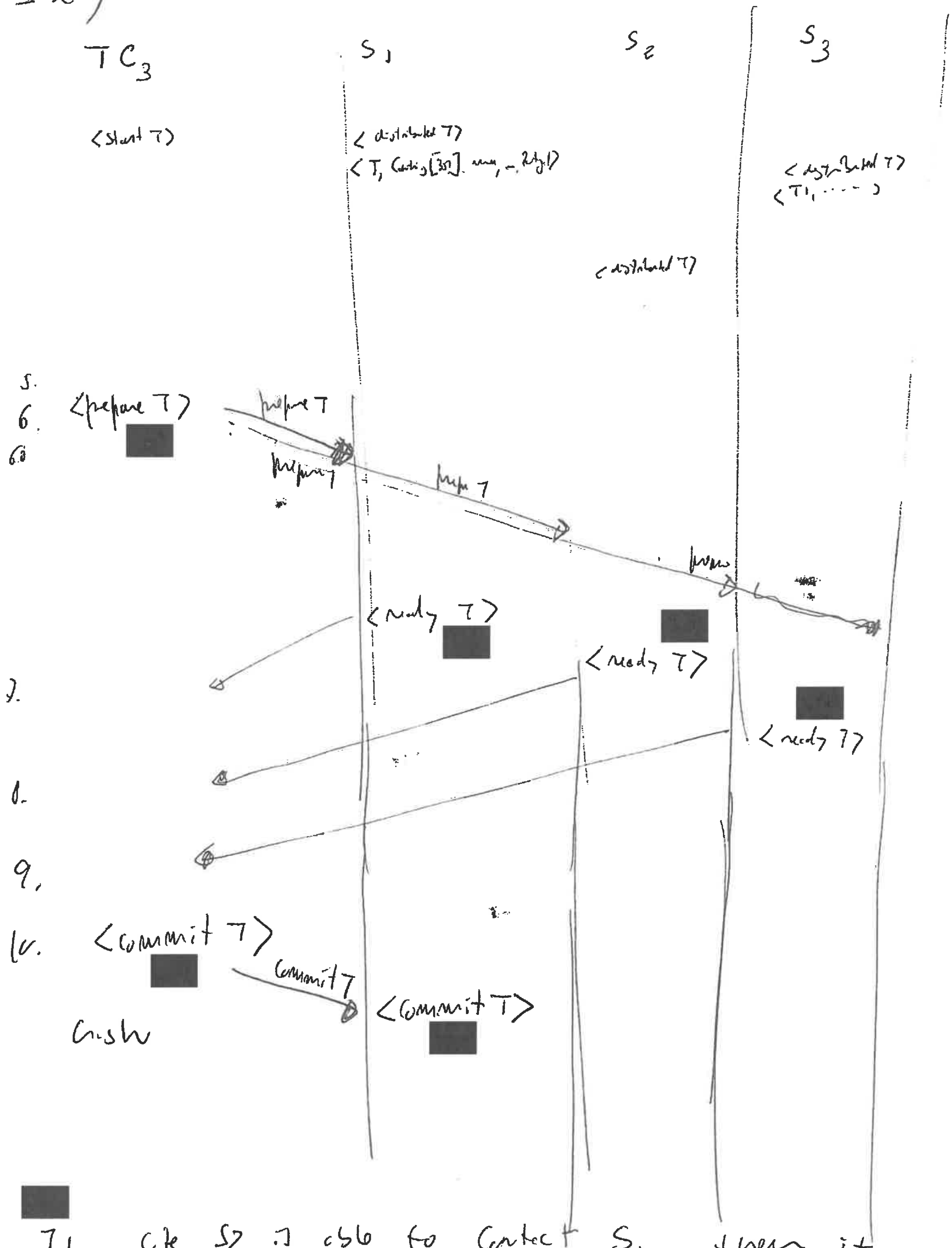
T2, T1, T3 [redacted]

T2, T3, T1

the value of N is the same as well as the content of table (content) [redacted]

M9

1e)



[redacted]

If site S2 is able to contact S1, then it can decide to commit. If not, it has to wait ~~tt~~ TC3 become online.

II

a)

we know that

$$\theta_1 \vee \theta_2 \vee \dots \vee \theta_m \equiv \neg(\neg\theta_1 \wedge \neg\theta_2 \wedge \dots \wedge \neg\theta_m)$$ [redacted]

So the factor would be $\simeq P(\neg(\neg\theta_1 \wedge \neg\theta_2 \wedge \dots \wedge \neg\theta_m)) =$

$$= 1 - P((\neg\theta_1) \wedge (\neg\theta_2) \dots \wedge (\neg\theta_m))$$ [redacted]

assuming independence ~~[illegible]~~ of the conditions

$$= 1 - P(\neg\theta_1) * P(\neg\theta_2) * \dots * P(\neg\theta_m)$$ [redacted]

$$= 1 - [1 - P(\theta_1)] * [1 - P(\theta_2)] * \dots + [1 - P(\theta_m)]$$ [redacted]

$$= 1 - \left(1 - \frac{s_1}{m_n}\right) * \left(1 - \frac{s_2}{m_n}\right) * \dots * \left(1 - \frac{s_m}{m_n}\right)$$

multiplying by the

$$m_n * \left[1 - \left(1 - \frac{s_1}{m_n}\right) * \left(1 - \frac{s_2}{m_n}\right) * \dots * \left(1 - \frac{s_m}{m_n}\right)\right]$$

25) Oracle implementation uses locks to detect potential write-write conflicts.
When a second transaction tries to ~~update~~ write a changed tuple previously updated by a concurrent transaction it will ~~[illegible]~~ wait.
[illegible] ~~[illegible]~~ when ~~[illegible]~~ decides to commit, the
[redacted] T2 would abort.

Comparison	FUW	FCW
Conflict Detection	on first update	On commit
wasted work	low	Higher
locking	yes [redacted]	NO [redacted]
[redacted] Pros	Early Conflict Detection	Higher Concurrency
	[redacted]	[redacted]
Cons	Less Concurrency	Potentially wasted work

Example: [redacted]

	T1	T2
[illegible]	begin	begin
	UPDATE	
		[redacted] [redacted]
		UPDATE ← waits in FUW after [illegible]
		COMMIT ← wins in FCW [redacted]
	COMMIT ← aborts in FCW [redacted]	

2c) log force is used to commit a transaction by forcing all its log records to stable storage.

This is necessary to apply when the log records are buffered [redacted] It can also be used with WAL when [redacted] data block pages need to be written into disk in order to maintain consistency of the log in stable storage and data stored in disk.