Recovery

1. Which of the following are true statements about buffer policies? (More than one may apply)

a. With a NO FORCE buffer policy, all pages dirtied by a transaction must be on disk before that transaction can commit. F

b. With a NO FORCE buffer policy, there might be the need for a REDO pass during recovery. T

c. With a FORCE buffer policy, the log record describing an update to a page must be forced to disk before the page itself. F

d. With a NO STEAL buffer policy, all pages dirtied by a transaction must be on disk before that transaction can commit. F

e. With a STEAL buffer policy, pages dirtied by a transaction might be on disk before that transaction commits. T

f. With a STEAL buffer policy, there might be the need for an UNDO pass during recovery. T

2. After a database crash, the system comes back online and you find the log and checkpoint information below on disk. For the following questions, consider the execution of the ARIES recovery algorithm. Assume a STEAL / NO FORCE buffer policy, and that no updates made it to disk after the checkpoint.

Log

LSN	Log Record	PrevLSN	UndoNextLSN
40	Update: T1 writes P1	Null	
50	Update: T2 writes P2	Null	
60	Update: T1 writes P1	40	
70	BEGIN CHECKPOINT		
80	T1 COMMIT	60	
90	T1 END	80	
100	Update: T2 writes P2	50	
110	END CHECKPOINT		
120	Update: T3 writes P3	Null	
130	Update: T2 writes P2	100	
140	Update: T3 writes P3	120	
150	T2 ABORT	130	
160	CLR: T2 UNDO 130	150	100

Transaction Table

Transaction ID	lastLSN	Status
T1	60	Running
T2	50	Running

Dirty Page Table

Page ID	RecLSN
P1	40

a. **What should the dirty page table and transaction table look like after analysis?**

TID	**lastLSN**	**Status**
T2	160	Aborting
T3	140	Running

PageID	**recLSN**
P1	**40**
P2	**100**
P3	**120**

b. **What log records will be read during redo? What log records will be redone?**
   Read: 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160
   Redone: 40, NOT 50, 60, 100, 120, 130, 140, 160

c. **What log records will be read during undo? What log records will be undone?**
   ToUndo = {160, 140} => {140, 100} => Undo 140 => {120, 100} => Undo 120 => {100} => Undo 100 => {50} => Undo 50 => Done.
   read: 160, 140, 120, 100, 50
   undo: 140, 120, 100, 50

d. **After ARIES completes, what new log records will have been written? Assume that LSNs start at 170 and increment by 10.**

LSN	Log Record	PrevLSN	UndoNextLSN
170	CLR: T3 UNDO 140	140	120
180	CLR: T3 UNDO 120	170	Null
190	T3 END	180	
200	CLR: T2 UNDO 100	160	50
210	CLR: T2 UNDO 50	200	Null
220	T2 END	210	

Query Optimization
Consider the following relational schema (with primary keys underlined):
STUDENTS (sid, s_name, street, city, age)
COURSES (cid, c_name, prof_name)
REGISTERED (sid, cid, credits)

where sid and cid are foreign keys referencing STUDENTS and COURSES respectively.
with the following characteristics:

- The STUDENTS relation has 40,000 tuples
- 20 tuples of STUDENTS fit in one disk page
- NKeys(city) for STUDENTS is 500 (i.e., there are 500 distinct values for city)
- NKeys(age) for STUDENTS is 50, and ages range from 10 - 60
- The REGISTERED relation has 60,000 tuples
- 50 tuples of REGISTERED fit in one disk page
- The COURSES relation has 800 tuples
- 40 tuples of COURSES fit in one disk page
- NKeys(prof_name) for COURSES is 20
- NKeys(c_name) for COURSES is 100

and the following indexes:

- A clustered B+Tree index is defined on the age attribute for STUDENTS (800 pages)
- An unclustered B+Tree index is defined on the city attribute for STUDENTS (800 pages)
- An unclustered B+Tree index is defined on prof_name for COURSES (10 pages)

For parts a, b, and c, state an efficient method for answering this query and state how many disk accesses will have to be made in order to answer the query using this method. Be sure to state which index(es) if any you are using. You may assume prof_name and age are uniformly distributed.

a) SELECT * FROM COURSES WHERE c_name = 'Database Systems' and prof_name = 'Brewer';

Access method: We have two competing plans here -
Scan of the COURSES relation (800/40 = 20 pages) vs. Unclustered hash index on prof_name + filter c_name on the fly. = (NPages(I) + NTuples(R))*RF = (10+800)*(40/800) = 40 - so we should just do the sequential scan.

Number of I/Os: 20

b) SELECT * FROM STUDENTS WHERE age > 30 AND sid = 1234567

Access method: index

Number of I/Os: 1680 = (800 + 2000)*(3/5) (vs 2000 for table scan)

c) SELECT * FROM REGISTERED R, COURSES C WHERE R.CID = C.CID;

For this query only, assume we have 10 pages of space in our Buffer Pool.

Join Method (circle one): [✓ CHUNK] HASH-JOIN SORT-MERGE

Outer Table (circle one): REGISTERED [✓ COURSES]

Number of I/Os:

Chunk nested loop join with courses as outer -

([R]/(B-2))*[S] + [R] = (1200/8)*20+1200 vs. (20/8)*1200+20 = 3020

Hash join - 3([R]+[S]) = 3600