Carnegie Mellon University
Department of Computer Science
15-415 - Database Applications
Spring 2000

FINAL EXAMINATION

Important points:

- **3 hours** duration
- **All aids** allowed (open books, open notes, calculators etc., except computer with network connection)
- Graded out of **100** points. Numbers in [**square brackets**] indicate points.
- You should have **9 non-empty pages, including this cover**
- **Always** highlight your final answer; unsolicited explanations will be used to give you partial credit, if your answer is wrong.
- When done, return **all** booklets, plus this handout.

LAST NAME (pls print)	
First Name	
andrew login	

Contents

Q1. Normal Forms [05 pts]

Consider the relation $R(A, B, C)$.

1. Is it possible that it may be in 3NF, but not BCNF?

   ANSWER: Yes [**2 pts**]

2. If yes, give some (simple) functional dependencies, so that R is in 3NF, but not BCNF. If not, explain, or point to a theorem in the book. [**3 pts**]

   ANSWER: $AB \rightarrow C$; $C \rightarrow A$

Q2. Storage - RAID [10 pts]

Compare RAID Level-3 (bit-interleaved parity) with RAID Level-5 (block-interleaved distributed parity), with respect to 'small reads', 'small writes', 'large reads' and 'large writes'. The goal is to have as high throughput as possible, that is, maximum number of transactions completed per second). For each setting,

1. choose among 'level-3 wins', 'level-5 wins', and 'tie' and mark your responses on the table of Figure Q2. Negative points for wrong answers. [**4 pts**]

2. justify your answers briefly [**6 pts**]

winner / Setting	Level-3	Level-5	Tie
small reads			
large reads			
small writes			
large writes			

Figure 1: RAID - mark your response with 'X'

ANSWER: sr: '5'; lr: tie; sw: '5'; lw: tie. For small reads, level-3 will engage all the disks for a single transaction, blocking the other transactions, while level-5 won't. Similarly for small writes; moreover, the single parity disk of level-3 will be the bottleneck, eliminating all parallelism among the writes. For large reads and writes, each transaction will block all the disks, for both RAID levels.

Q3. Indexing - I [07 pts]

Consider the relation $EMP(ssn, name, salary)$, with the following specifications:

- it has ssn as the primary key,

- it has n tuples, each 50 bytes long,
- there are many, *exact match* queries on the *ssn* only
- there are rare insertions and deletions.

Assume a page (=block) size of 8Kb, and consider the indexing alternatives:

- (a) no index
- (b) B-tree clustering index on *ssn*
- (c) B-tree non-clustering index on *ssn*
- (d) a hashed index on *ssn*

If you were the DBA, which indexing alternative you would choose

1. if n=100 tuples [**1 pts**] Justify your answer briefly [**1 pts**]

   ANSWER: 'no index', because the whole table fits in a block. 1 disk access and sequential scan will be very fast anyway

2. if n=1,000,000 tuples [**2 pts**] . Jystify briefly [**3 pts**] .

   ANSWER: 'hashed index', because it is the fastest, and none of the disadvantages of hashing will create a problem: there is no growth/shrinkage of the table; there is no need for maintaining the key ordering, since we only have exact match queries

Q4. Indexing -II ________________________ [10 pts]

Consider B-trees of order 5, that is, there are at most 5 pointers per node. What is the most sparse such B-tree we can ever have, that stores the integers $1, 2, \ldots, 17$.
ANSWER: 3 levels: '9' is the only key at the root, the left parent has keys '3' and '6', the right parent has keys '12' and '15', and the leaves have each two keys: (1,2), (4,5) ... (16,17)

Q5. Query optimization - Join plans ________ [10 pts]

Consider the join of relation 'R' with relation 'S', where 'R' spans $b_r = 1,000$ pages and 'S' spans $b_s = 100$ pages. Let k be the number of buffers (=pages) that we have in main memory, for the blocked nested loop method. Also assume that we do **not** scan relations backwards. Recall that the 'outer' relation is the one in the outer loop of the nested loop method.

1. for k=2 buffers,

   - which relation should be the outer relation? [**1 pts**]
     ANSWER: the one with the fewest pages: 'S'

- how many disk accesses we shall need then? [**3 pts**]
  ANSWER: $b_s + b_s * b_r$ *= 100 + 100,000 = 100,100*

2. for k=200 buffers,

   - which relation should be the outer? [**1 pts**]
     ANSWER: again, 'S'
   - how many buffers should we give it? [**2 pts**]
     ANSWER: b_s*=100 buffers - the maximum possible to keep it all in main memory*
   - how many disk accesses we shall need then? [**3 pts**]
     ANSWER: $b_s + b_r$ *= 1,100 - just a single scan of each table*

Q6. Query optimization - selectivities ____ [10 pts]

Consider the relation $LEG(source, destination)$, recording non-stop flight-legs from one airport ('*source*') to another ('*destination*'). Assume that it contains n = 1,000 tuples. Consider the query to find all airports within two hops from Pittsburgh (keep the duplicates, for simplicity):

```
select L2.destination
from LEG as L1, LEG as L2
where L1.destination = L2.source and
      L1.source = 'Pittsburgh'
```

Estimate the number of output tuples for the above query if we have 100 distinct airports, that is $V(LEG, source) = V(LEG, destination) = 100$. *Hint: First, estimate the number of tuples in the self-join*
ANSWER: 100 tuples. We have $n_r * n_r$ */ V(LEG, source)= 1000 * 1000/100 tuples for the join, that is, all the pairs of airports that are within 2 hops; and then we need to divide by V(LEG, source)=100 for the ones originating from 'pittsburgh'*

Q7. Recovery ____ [21 pts]

Figure 2 shows three logs of three different DBMSs, each after a crash. The logs respectively operate under (a) the deferred update scheme, (b) the incremental update scheme with checkpoints, and (c) the incremental updates scheme, without checkpoints,

For the incremental updates, recall that the format of each log record is

(transaction-id, item-id, old-value, new-value)

For each of the three cases, answer the following questions - there will be **negative** points for wrong answers:

(T1 start)	(T50 start)	(T51 start)
(T2 start)	(T50, K, 10, 20)	(T51, X, 10, 20)
(T1, A, 50)	(T60 start)	(T61 start)
(T2, B, 100)	(T50 commit)	(T51 commit)
(T3 start)	(T60, L, 100, 200)	(T61, Y, 100, 200)
(T2 commit)	(T70 start)	(T71 start)
(T3, C, 500)	(T70, M, 1000, 2000)	(T71, W, 1000, 2000)
(T3, D, 600)	(checkpoint {T60, T70})	
(T4 start)	(T70 commit)	(T71 commit)
(T4, E, 900)	(T80 start)	(T81 start)
(T4 commit)	(T80, N, 750, 850)	(T81, Z, 750, 850)
–crash––	(checkpoint {T80, T60})	
	(T80 commit)	(T81 commit)
	–crash–––-	–crash–––-
(a) def. upd	(b) incr. upd	(c) incr. - no ckp

Figure 2: Write-ahead logs after a crash, with (a) deferred updates (b) incremental updates (c) incremental updates without checkpoints

1. just after the crash, and **before** we start the recovery algorithm, what are the values of the corresponding data items on the disk (A, B, …, K, L, M, …, X, Y, …)? The acceptable answers are: 'its old value', 'its new value', and 'can-not-tell' [**3×3 pts**]

   ANSWER: (a) def. updates: A,C,D: old; B,E: can-not-tell. (b) incr. upd.: all: new; (c) incr-no-ckp: all: can-not-tell:w

2. List the transactions that have to be undone (if any) [**3×1 pts**]

   ANSWER: (a): undo nothing; (b) undo T60; (c) undo T61

3. List the transactions that have to be redone (if any) [**3×1 pts**]

   ANSWER: (a): redo T2, T4; (b) redo T80; (c) redo T51, T71, T81

4. List the values of all the data items (A, B, ...), **after** the recovery algorithm is over. Again, the answers should be 'old', 'new', 'can-not-tell' [**3×2 pts**]

   ANSWER: (a) def. upd: A,C,D: old; B,E: new; (b) inc. upd: K,M,N: new; L:old; (c) inc. upd w/ckp: X,W,Z: new; Y:old.

Q8. Serializability ____________________ [05 pts]

Consider the schedule of Figure 3.

1. Is it serializable? If yes, give an equivalent serial execution; if not, explain briefly. [**1 pts**]

	T1	T2	T3
t9	...	...	...
t10	read(A)		
t11	write(A)		
t12	...	...	...
...			
t19	...	...	...
t20		read(A)	
t21		write(A)	
t22	...	...	...
...			
t29	...	...	...
t30			read(B)
t31			write(B)
t32	...	...	...
...			
t39	...	...	...
t40		read(B)	
t41		write(B)	
t42	...	...	...

Figure 3: An interleaved schedule

ANSWER: Yes: (T3, T1, T2); also correct (T1, T3, T2)

2. Is it at all possible that the above schedule is produced by the 2PL protocol? [**1 pts**]

   ANSWER: yes

3. If it is not possible to be produced by 2PL, explain briefly; if yes, show where the 'lock()' and 'unlock()' requests would be in time (you may mark on the figure your final answers, **cleanly**). [**3 pts**]

   ANSWER: the lock requests of T1 and T3 are obvious; for T2, we have: t19-lock(A); t39-lock(B); t42-unlock(A) unlock(B)

Q9. Locking - multiple granularity ________[10 pts]

Consider the multiple-granularity locking algorithm, operating on the following lock-able items:

1. 'db': the whole database
2. ACCOUNT: a table with attributes (account-id, customer-id, balance)
3. CUSTOMER: a table with attributes (customer-id, name, address)

4. $a_1, \ldots, a_{1000}$: the 1000 records of the ACCOUNT table

5. $c_1, \ldots, c_{100}$: the 100 records of the CUSTOMER table

Consider also the following transactions - specify **all** the locks that each will ask for, on what items, and with what order. For example, a (possibly wrong) answer for T1 could be: "*T1 will first ask for an 'IS' lock on ACCOUNT and then for X-locks on a_{10} and a_{50}.*

- T1: report the balance of two accounts, a_{10} and a_{50} [**3 pts**]

  ANSWER: IS for 'db'; IS for 'ACCOUNT'; S for a_{10} and a_{50}

- T2: increase all balances by 3% [**4 pts**]

  ANSWER: IX for 'db'; X for 'ACCOUNT'

- T3: move 10 dollars from a_{30} to a_{31} [**3 pts**]

  ANSWER: IX for 'db'; IX for 'ACCOUNT'; X for a_{30}; X for a_{31}

Q10. Deadlocks [02 pts]

Assume that we only have eXclusive locks, for simplicity. Consider the following transactions, T1, T2, T3, and T4, who have issued the following lock requests:

T1: lock(A); T2: lock(B); T3: lock(C); T1: lock(B); T4 lock(B);

1. Do we have a deadlock? [**1 pts**]

   ANSWER: No

2. Justify your answer. [**1 pts**]

   ANSWER: there is no cycle in the wait-for graph: T1 waits for T2 because of B; T4 waits for T2 because of B.

Q11. Semijoins [10 pts]

Consider the relations $R(A, B)$, $S(B, C)$ and $T(C, D)$, residing in three different sites. Suppose that each attribute is 4 bytes long.

R

A	B
a1	b1
a2	b3
a3	b5

S

B	C
b3	c1
b5	c2
b7	c5

T

C	D
c1	d3
c5	d20
c3	d50

1. Show the result of $R \ltimes S$ **[1 pts]**
   ANSWER: { *(a2,b3), (a3,b5)* }
2. What is the cost of the previous operation (in number of bytes transmitted)? **[1 pts]**
   *ANSWER: 3*4 = 12 bytes*
3. Show the result of $S \ltimes T$ **[1 pts]**
   ANSWER: { *(b3,c1), (b7, c5)* }
4. What is the cost of the previous operation (in number of bytes transmitted)? **[1 pts]**
   *ANSWER: 3*4 = 12 bytes*
5. Show the result of $R \ltimes (S \ltimes T)$ **[3 pts]**
   ANSWER: { *(a2,b3)* }
6. What is the total cost of these operation (in number of bytes transmitted)? **[3 pts]**
   ANSWER: 12+8=20 bytes

This is the end of the exam questions. ________________________________ Good luck!