Name _____

Do not write in this space

Class Account _____

UNIVERISTY OF CALIFORNIA, BERKELEY
College of Engineering
Department of EECS, Computer Science Division

CS186 J. Hellerstein
Spring 2003 Final Exam

Final Exam: Introduction to Database Systems

This exam has seven problems, each worth a different amount of points. Each problem is made up of multiple questions. You should read through the exam quickly and plan your time-management accordingly. Before beginning to answer a question, be sure to read it carefully and to *answer all parts of every question!* Please **do not spend time explaining your answers unless we explicitly ask that you do so**. We will not be giving extra or partial credit for explanations unless we ask for them.

You **must** write your answers on the exam. You also must **write your name** at the top of **every page**, and you must turn in all the pages of the exam. **Do not** remove pages from the stapled exam! Extra answer space is provided in case you run out of space while answering. If you run out of space, be sure to make a "forward reference" to the page number where your answer continues.

Good luck!

Name _____

1. **Recovery [20 points]**
   In this question we explore the interplay between buffer management and recovery. We present four scenarios below. For each scenario, you must follow the following guidelines:
   — Choose a maximally ***efficient*** protocol!
   — Among all protocols with the same level of efficiency, choose the ***simplest***.

   Each scenario corresponds to exactly one of the four quadrants in the chart at the bottom of the page. **Place the letter of each scenario (a, b, c, d) in the appropriate quadrant.** Note that while each letter goes in only one quadrant, it is possible that some quadrants will have more or less than one letter in them! **[5 points each]**

   a. **Scenario: Traditional ARIES.** You have been told to implement the ARIES recovery protocol as discussed in class.

   b. **Scenario: Buffer Pool Bigger than Database.** Modern servers ship with as much as 32GB of RAM. Hence for many customers, their databases will never be larger than available RAM. You have a startup company that is designing a special-purpose DBMS for these customers.

   c. **Scenario: Before-Image Logging.** In order to keep your logs more compact, you store only the "before-image" of the data in update log records, and not the "after-image". (Recall that the before-image has the state of the page before the update; the after-image would have also shown the state of the page after the update.)

   d. **Scenario: Untrusted Buffer Manager.** No matter how many times you explain it, the engineers implementing the buffer manager cannot understand the log manager. So you're going to have to choose a protocol that can handle any mix of page replacement decisions.

Force		
No Force		
	No Steal	Steal

Name _____

2. **ARIES [20 points]**
Consider the following log where the DPT represents the Dirty Page Table and TT represents the Transaction Table

LSN	Contents	prevLSN	undonextLSN
10	Update **T1** writes **p1**		
20	Begin Checkpoint		
30	End Checkpoint DPT: **(1,10)** TT: **(T1, Running, 10)**		
40	Update: **T1** writes **p3**	10	
50	Insert: **T2** writes **p5**		
60	Update: **T1** writes **p3**	40	
70	Update: **T2** writes **p8**	50	
80	Abort: **T1**	60	
90	CLR: **T1** Undo p3 LSN 60	80	40
100	Commit: **T2**	70	
110	End **T2**	100	
	CRASH, RESTART		

Answer the following questions:

a. What is the smallest LSN accessed during the Analysis phase. **[2 points]**

b. Fill in the contents of the **Dirty Page Table** and the **Transaction Table** at the end of the Analysis phase. (you may not need all the space we give you) **[10 points]**

PageID	RecLSN

XID	Status	LastLSN

c. At which LSN does the Redo phase begin? **[2 points]**

d. What entries (specify only LSNs) *do* get undone as part of the Undo phase? **[6 points]**

Name _____

3. **Functional Dependencies and Normalization [20 points]**
Consider the relation schema R = (A, B, C, D, E, F) and the set of functional dependencies F: **A->B, A->C, BC->E, BC->D, E->F, BC->F**

Note that in all the following, you will *never* have to compute F+, the closure of F!

a. List the *minimal* candidate key(s) for R. Write 'none' if you think there are no candidate keys. **[2 points]**

b. List the FDs in F that violate BCNF. (Hint: There are four) **[4 points]**

c. Is R in 3NF (yes or no)? **[2 points]**

d. Is F a minimal cover? **[2 points]**

(continued)

e. Suppose we decompose R into the following tables:
R1 = (B, C, E)
R2 = (B, C, F)
R3 = (B, C, D) and
R4 = (A, B, C).
This decomposed schema is indeed in BCNF (you can trust us on this!)

Unfortunately, this decomposition is not dependency-preserving; in particular, the dependency E->F cannot be checked on a single table. A CHECK ASSERTION can be used to enforce E->F.

Complete the following SQL statement for this particular CHECK ASSERTION needed to guarantee E->F. **[8 points]**

```
CREATE ASSERTION checkDep
CHECK ( NOT EXISTS
( SELECT * FROM  R1, R2

WHERE _____

GROUP BY _____

HAVING COUNT(_____)_____))
```

f. Why might the ASSERTION in (e) be expensive? **[2 points]**
   i. Updates to R1 and R2 are frequent
   ii. Inserts to R1 and R2 are frequent
   iii. Insertions to R2 are frequent; R1 rarely changes.
   iv. Reads to R1 and R2 are frequent
   v. (i) and (ii)
   vi. (i), (ii), (iii)
   vii. All of the above

Answer (choose *one*): _____

Name _____

4. **Database Tuning [14 points]**
Consider the following BCNF relational

Apartment (aid, capacity)
GraduateStudent (SID, age, sex, dept, GPA, aid)

— GraduateStudent.aid is a foreign key referencing Apartment.
— There are very few "single" apartments (i.e. where capacity=1)
— 50% of the Graduate Students are males.

a. You are told that the following three queries are the most common. For each of these queries, we have listed three different indexes as possible access methods. Pick the access method that would benefit the query **most** in terms of I/O performance. Assume that these queries occur much more frequently than updates. **[3 points each]**

Query 1: List aids of apartments with capacity=1
i. Clustered B+Tree index for Apartment on aid field
ii. Unclustered B+Tree index for Apartment on capacity field
iii. Clustered B+Tree index for Apartment on capacity field.

Answer (choose *one*): _____

Query 2: List SIDs of male graduate students
i. No indexes. Use a file scan on GraduateStudent.
ii. Unclustered B+Tree index for GraduateStudent on sex
iii. Clustered B+Tree index for GraduateStudent on SID

Answer (choose *one*): _____

Query 3: List depts of graduate students staying in apartment aid=4.
i. Clustered B+Tree index for GraduateStudent on aid
ii. No indexes. Use a file scan on GraduateStudent table.
iii. Unclustered B+Tree index for GraduateStudent on <aid, dept>

Answer (choose *one*): _____

continued

b. The database is still running very slowly even after you recommended the right indexes in part (a)! A careful workload study reveals that in practice, the following two transactions are *extremely* frequent (far more than any other queries or updates):

- **T1:** Find the average age of all graduate students group by aid
- **T2:** Update the GPA of graduate student *g* (the value of *g* may be different each time T2 is run)

Traces show that the above two queries resulted in concurrency control bottlenecks, due to lock contention on the GraduateStudent table. Tuple granularity locks are obtained.

The following solutions have been proposed to alleviate the problem. Choose the **single** letter that gives the most correct options; if no correct options are listed, choose (ix). **[5 points]**

i. Vertically partition (i.e. decompose) the GraduateStudent table into two separate tables such that (SID,age,aid) is in one table, and (SID, sex, dept, GPA) is in another.
ii. Horizontally partition the GraduateStudent table according to aid field. Graduate Students who stay in even numbered apartments are stored in a different table from those that are odd numbered.
iii. Make use of page granularity locks
iv. Use non-strict two-phase locking (2PL)
v. Create an unclustered B+Tree index for GraduateStudent on <aid, age>
vi. Either (i) or (ii) would work
vii. Either (i) or (v) would work
viii. Either (iii) or (iv) would work
ix. None of the above.

Answer (choose *one*): _____

Name _____

5. Query Optimization [15 points]

Consider the following relational schema and SQL query:
Student (SID, DID, Enroll_Year, Nationality)
Department (DID, Name, Building_Num, Telephone, FID)
Finance (FID, Budget, Expenses, …)

```
SELECT D.Name, F.Budget
FROM Student S, Department D, Finance F
WHERE S.DID = D.DID and D.FID = F.FID AND
      D.Building_Num > 5 AND D.Building_Num <=10 AND
      S.Enroll_Year = 2000 OR S.Enroll_Year = 2001;
```

Here are some statistics:
— Building Numbers range from 1 to 20 inclusive (i.e. 1 and 20 are both valid numbers).
— Each building has the same number of departments
— Students' enrollment year (Student.Enroll_Year) ranges from 1997 to 2002, and is distributed according to the following table:

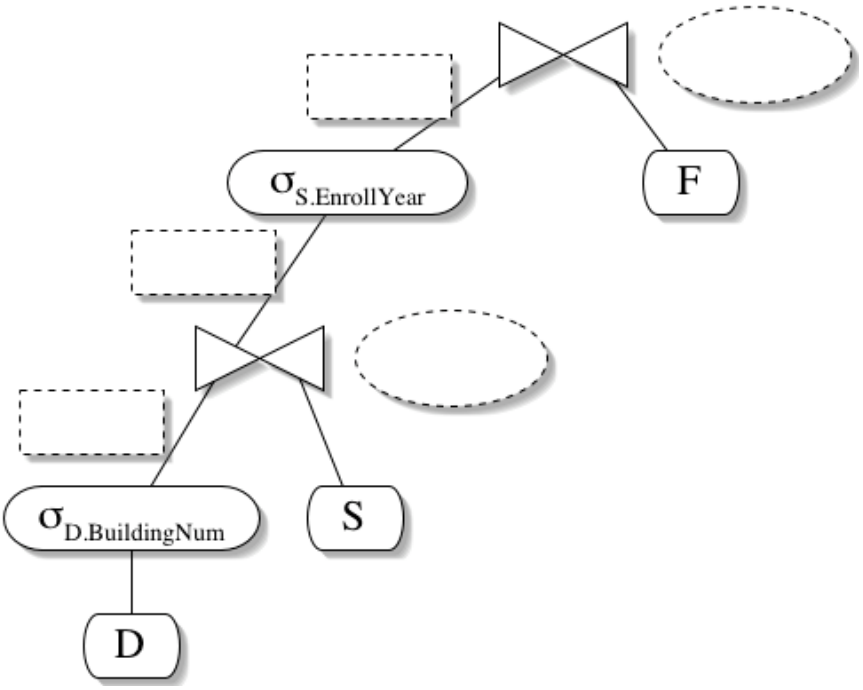
1997	1998	1999	2000	2001	2002
5000	1400	2000	3000	7000	1600

— Number of Tuples (pages) per relation:
- Student: 20000 (2000 pages)
- Department: 100 (10 pages)
- Finance: 100 (10 pages)

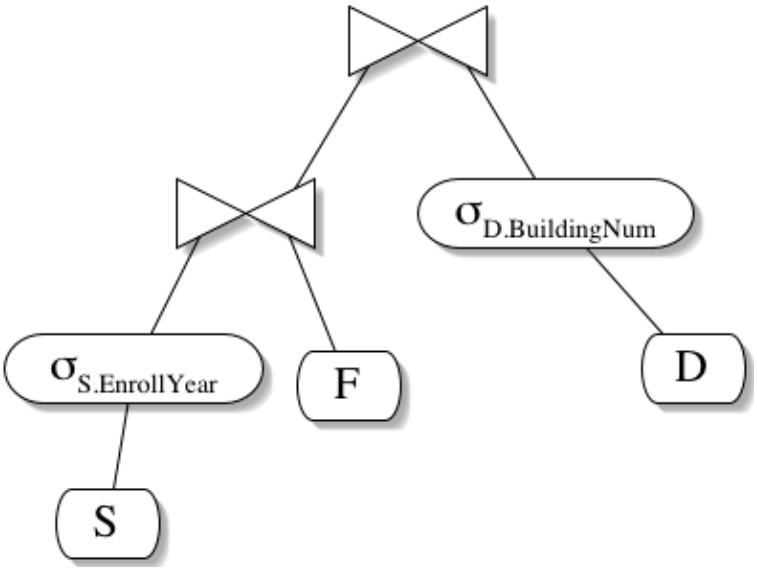
(continued)

Name _____

a) Consider the following query plan. Each of the leaf nodes is a *sequential scan*. Each of the joins is a *naïve nested loops join*. In the ***three dotted boxes***, write down the expected number of tuples generated by the subtree below the associated tree edge. In the ***two dotted ovals,*** write down the expected number of I/Os performed by the associated join. (You can assume that projections are done as early as possible.) **[12 points]**



b) Would a query optimizer like the one we studied in class choose the following plan? (Again, assume that projections are done as early as possible.) ***Briefly*** explain your answer! **[3 points]**



Name _____

6. Concurrency control [16 points]

a) Consider the following modified definition of serializability:
A schedule S is serializable iff it produces the same database state as a serial schedule T, where the transactions in T are exactly those in S, and are ordered in T according to their first appearance in S.
Is this definition: **[3 points]**

i. Correct?
ii. Overly restrictive: i.e. there are some serializable schedules not covered by this definition?
iii. Overly permissive: i.e. there are some *unserializable* schedules that are covered by this definition?
iv. (ii) and (iii)

Answer (choose *one*): _____

b) Bob and Anne share a bank account for their business. Today they went to the bank at the same time. Draw the dependency graph for the schedule below. You do not need to label any edges in the graph. **[3 points]**

T_Bob

Description	Action
Looks at checking balance	R(C)
Withdraw $100 from checking	W(C)
	commit

T_Anne

Description	Action
Looks at savings balance	R(S)
Looks at checking balance	R(C)
Transfer $200 from checking to savings	W(C)
	W(S)
	commit

c) Is the schedule in (b) conflict serializable? If so, give an equivalent serial schedule. If not, enumerate all the serial schedules, and explain how Bob and Anne's experience would be changed in each. **[5 points]**

d) Suppose that the database system at the bank implemented strict 2-phase locking as we studied in class. Assume that Bob and Anne's requests for actions arrive in the same order as in (b), but if either of them is blocked while waiting for a lock, their actions stop arriving until they acquire the lock (after which time they continue as fast as they can). Describe what happens in that scenario: a few words should suffice. **[5 points]**

Name _____

7. **Sorting and Hashing [12 points]**
Consider the following query:

SELECT g, COUNT(*) FROM T GROUP BY g

Assume that table T is of size N disk blocks, and the buffer pool has (B+1) frames available for this grouping operation.

NOTE: We assume that N is a little ***more*** than B+1. (Think about this!)

We will consider both hash-based and sort-based implementations of grouping. For hash-based grouping, we will use the scheme you implemented for homework 2. For sort-based grouping, we will use the scheme described in class, with ***heapsort*** (also known as ***tournament sort***) as our in-memory sorting algorithm.

Note that we are ***not*** writing out the final result of sorting or hashing, but we ***do*** include the cost of initially scanning table T!!

For each technique in the table below, write down the letter of the correct cost solution (a – f). Each technique has exactly one matching cost, but a given solution may appear 0, 1 or many times in the table. **[3 points each]**

Possible solutions include the following costs:
a) N I/Os
b) 3N I/Os
c) 5N I/Os
d) 6N I/Os
e) 7N I/Os
f) B I/Os

Technique	Solution
Sort-based, average case	
Sort-based, worst case	
Sort-based, best case	
Hash-based, best case	

Name _____

SCRATCH

Name _____

SCRATCH