Prof. Brewer Fall 2015 — CS 186 Databases — Midterm 3

PRINT your name: _____, _____
(last) (first)

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct.

SIGN your name: _____

PRINT your class account login: `cs186-`_____ and SID: _____

Your TA's name: _____

Your section time: _____

Login of the person sitting to your left: _____ Login of the person sitting to your right: _____

You should receive a double-sided answer sheet in addition to this packet. Mark your name and login on the answer sheet, and in the blanks above. **All answers must be marked on the answer sheet – do not show any work unless asked.** You have 80 minutes for this exam – do not spend too much time on any single problem. Turn in both this packet and the answer sheet when you are done.

Do not turn this page until your instructor tells you to do so.

Question:	1	2	3	4	Total
Points:	20	23	15	27	85
Score:					

Problem 1 ***SQL and Query Optimization*** **(20 points)**

For this problem, show your work for partial credit, and leave all numerical answers as simple expressions. We have provided plenty of room on the answer sheet, and some lines/whitespace may be unnecessary.

As an aspiring paleontologist you decide to explore the exciting world of fossils while putting your databases knowledge to good use. You recall that our very own Campanile has a fine collection of fossils (numbering in the hundreds of thousands!), and lo and behold, you find the tables:

1. **Collection**, which has a record for every fossil stored in the Campanile,
2. **Donors**, which has a record for every donor that donated to the collection, and
3. **Fossils**, which contains the sum total of all human knowledge on fossils.

Collection	[C] = 2^{13} $p_C = 2^5$	**Donors**	[D] = 2^7 $p_D = 2^6$	**Fossils**	[F] = 2^{17} $p_F = 2^8$
id	int	id	int	id	int
donor_id	int	num_donated	int	age	int
floor	int *[2-7]**	donor_since	int	type	String
condition	int *[1-100]**	...		...	

** = inclusive range of values, for which you may assume a uniform distribution*

A few other notes:

- We shorthand Collection as C, Donors as D, Fossils as F.
- C.donor_id is a foreign key referencing D.id.
- The ids in C are strictly a subset of the ids in F.

You also have a few indexes, which may or may not be useful:

1. A clustered index on F.id [2^{10} pages, height = 3]
2. An unclustered index on F.age [2^8 pages, height = 1]
3. A clustered index on (D.donor_since, D.num_donated) [2^3 pages, height = 1]

---

1. **[6 points]** You want to find the average condition of fossils in the Campanile collection, grouped and sorted by age in ascending order. However, you only want to look at fossils on **floors 5, 6, and 7**, and from small-scale donors (**fewer than 100 fossils donated**) who have **not donated prior to the year 2000**. Write a SQL query that expresses this.

You then want to prove to the world (or just us) that you know how a System R query optimizer estimates plan costs for the SQL query above (**you can do this problem using only the description from Q1). In addition to what you already know from the query, you estimate that **roughly half of donors have donated fewer than 100 fossils**, and roughly **1/8th of donors started donating from 2000 or later**.

2. [**7 points**] Naturally, you begin by considering **single table plans**.

   (a) For each table, calculate:

   - the IO cost of a file scan
   - the number of pages that will be passed down the pipeline ("downstream")

   (b) Name **all** non-filescan single-table plans that are kept (be specific). Choose **one** plan (indicate your choice), and calculate its estimated cost. *Recall that System R optimizers will consider interesting orders.*

3. [**7 points**] You recall that System R optimizers never consider cross joins given alternatives, so you narrow your **two-table plans** to $C \bowtie F$, $F \bowtie C$, $C \bowtie D$, and $D \bowtie C$.

   (a) If we were to use Chunk-Nested Loop Joins for the four joins listed above, find the **smallest** and **largest** reduction in IOs that would result from a single pre-written temp file. *Think through this problem before attempting any calculations.*

   (b) Calculate (as a System R query optimizer) the cost of an Index-Nested Loop Join for $C \bowtie F$. Do include costs for reading C (the "outer" loop).

   (c) How many tuples will be passed down the pipeline for this join (Index-Nested Loop Join between $C \bowtie F$)? For a Chunk-Nested Loop Join?

Problem 2 ***Concurrency Control*** **(23 points)**

1. [**14 points**] Consider the following schedule S. Assume that each transaction commits or aborts immediately after its last operation, and that locks can be upgraded. For example, T1 aborts or commits immediately after timestamp 5.

time	1	2	3	4	5	6	7	8	9	10	11	12
T1	R(A)	R(B)	W(B)		R(C)							
T2							R(B)	W(B)			R(C)	
T3				R(A)		R(B)			W(A)	W(C)		R(D)

   (a) Write the number of conflicting operations in S.

   (b) Draw the dependency graph for S.

   (c) Select all schedules below that are conflict equivalent to S.

i.

T1	R(A)		R(B)	W(B)	R(C)							
T2								R(B)	W(B)	R(C)		
T3		R(A)				R(B)	W(A)				W(C)	R(D)

ii.

T1	R(A)	R(B)		W(B)	R(C)							
T2							R(B)		W(B)			R(C)
T3			R(A)			R(B)		W(A)		W(C)	R(D)	

iii.

T1	R(A)		R(B)			W(B)	R(C)					
T2					R(B)						W(B)	R(C)
T3		R(A)		R(B)				W(A)	W(C)	R(D)		

iv.

T1	R(A)	R(B)	W(B)	R(C)								
T2										R(B)	W(B)	R(C)
T3					R(A)	R(B)	W(A)	W(C)	R(D)			

   (d) Write the timestamps of one possible set of operations that can be removed from S such that the resulting schedule adheres to the strict 2-phase locking protocol. **Use the minimum number of operations possible.**

   (e) Does S avoid cascading aborts? If not, state one transaction that would cause a cascading abort if aborted, and which operation can be removed from the transaction to avoid that cascading abort.

2. [**5 points**] Consider a database where A is a table in the database and contains pages B and C. Page B contains tuples D and F, and page C contains tuples G and H. Assume you have the following locking schedule for transactions T1, T2, T3, and T4 that utilizes multiple granularity locks. Assume no locks are released within the given timeframe, and no other transactions are running.

time	1	2	3	4	5	6	7	8	9	10	11	12
T1	IX(A)				SIX(C)			____(B)			____(G)	
T2		IS(A)		IS(B)		S(D)						____(C)
T3			IX(A)				IX(B)		____(D)	____(F)		

(a) Write all possible locks that the given transaction can acquire on the given object **without having to wait for another transaction**.

i. At time 8

ii. At time 9

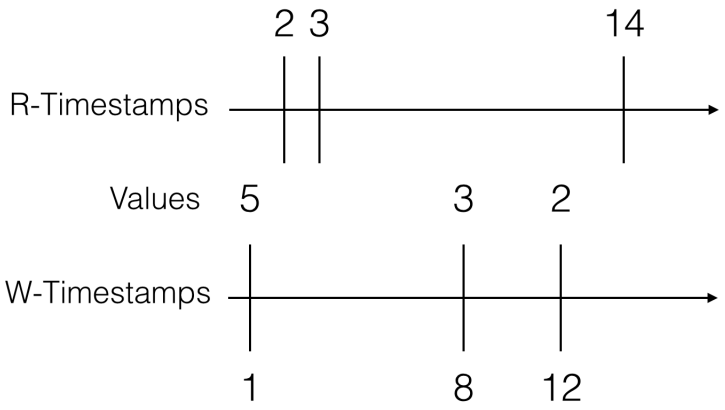
iii. At time 10

iv. At time 11

v. At time 12

(b) Provide a minimal set of locks from timestamps 8 through 12 that would force a deadlock in the schedule. If no lock at a particular timestamp affects deadlock, write "None".

3. [**4 points**] Consider the following Timestamp Ordered Multi-version concurrency control timeline for a tuple X.



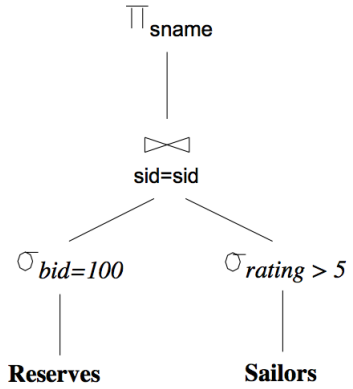
(a) What value of X would a transaction with timestamp 13 read?

(b) What value of X would a transaction with timestamp 7 read?

(c) At which timestamps would an attempted write result in an abort?

Problem 3 ***Iterator Model.*** **(15 points)**

Given the following query plan, answer the following questions on the answer sheet:



On the Appendix for the Iterator Model, there are method signatures and docstrings provided. Those are all the functions you're expected to use. There are implementation details left out.

Here are a few tips and assumptions:

- The single table access plans are file scans (SeqScan)
- Specific operator predicates will not matter.
- This query will run to completion without interruption.
- Exceptions thrown are not important
- `Join` is an implementation of a Nested Loop Join.

---

1. After calling `Query.execute()`, what order will the `open()` calls be made? Fill out the answer sheet. We are only considering the calls made before we start iterating through the tuples. You may or may not use all the lines provided.

   Leave all answers in form of (class).open(); ie, `Project.open()`. You may ignore `Query.start()`.

2. In a proper, working implementation of SimpleDB, between calls of `Query.next()`, which of the following could be called? Circle on answer sheet.

   (a) `Filter.rewind();`
   (b) `Filter.open();`
   (c) `Join.open();`
   (d) `SeqScan.open();`
   (e) `Filter.close();`
   (f) `Filter.next();`
   (g) `Join.next();`
   (h) `SeqScan.next();`

Problem 4 ***Logging and Recovery*** **(27 points)**

Please see Appendix: Recovery to find the log in reference. This is the log found on disk as we begin the recovery process. If it is attached to your exam, you may rip it off.

1. [**3 points**] Fill in the missing records denoted by "???". Please follow the same format as the records provided.

The system comes back up after the crash and starts performing recovery.

2. [**4 points**] Fill in the transaction table as it appears at the end of analysis. You may not need to use all rows in the table.

3. [**4 points**] Which pages are in the dirty page table at the end of analysis? You may not need to use all rows in the table.

4. [**5 points**] What actions are redone during the REDO phase? List the LSNs on the answer sheet in the order that they are redone, separated by commas. No new log records will be written during REDO. You do not need to include actions that do not change pages.

5. [**7 points**] What are the records written during the UNDO phase? Start writing new log records at LSN 240. The difference between the LSNs of two adjacent log records should be 10. You may not need to use all rows in the table.

6. [**4 points**] In the ARIES protocol, recovery consists of 3 distinct, sequentially executed phases: Analysis, REDO, and UNDO. However, in hw5, we were able to perform Analysis and REDO operations at the same time, in a single pass of the log. Briefly explain the key difference between ARIES and SimpleDB that allowed us to do this.