University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Fall 2009

Prof. Michael J. Franklin

MIDTERM I SOLUTIONS

CS 186 Introduction to Database Systems

NAME:_____ **STUDENT ID:**_____

IMPORTANT: Circle the last two letters of your class account:

cs186 a b c d e f g h i j k l m n o p q r s t u v w x y z

a b c d e f g h i j k l m n o p q r s t u v w x y z

DISCUSSION SECTION DAY & TIME:_____ **TA NAME:** _____

This is a **closed book** examination – but you are allowed one 8.5" x 11" sheet of notes (double sided). You should answer as many questions as possible. Partial credit will be given where appropriate. There are 100 points in all. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time-consuming than others.

Write all of your answers directly on this paper. **Be sure to clearly indicate your final answer** for each question. Also, be sure to state any assumptions that you are making in your answers.

GOOD LUCK!!!

Problem	Possible	Score
1. Formal Query Languages	25	
2. Buffer Manager	30	
3. Hash Indexes	20	
4. Tree Indexes	15	
5. Short Answer	10	
TOTAL	100	

SID:_____

Question 1 – Formal Query Languages [4 parts, 25 points total]

Through nefarious means, you managed to get your hands on a copy of the EECS instructional database (with the following schema, primary keys are underlined).

Student (sid, sname, address, age)

Course (cid, cname, credits)

Professor(pid, pname, address, age, department)

Grade(sid, cid, pid, grade)

Eyeing a juicy scholarship, you want to use this information to improve your gpa. You'll need to write queries to get the data you need. Note: we don't care about efficiency, but **points will be taken off for using unnecessary relations in your queries**. Answer each question using both the relational algebra and the relational calculus.

a) [8 points (4 each)] First, you'll need to take care of your competition. Write a query that returns the all students who have received an A in a class taught by Professor Hilfinger.

i. Relational Algebra

$$\Pi_{sid,sname,address,age}(\sigma_{grade=A \wedge pname=Hilfinger}(\text{Student} \bowtie \text{Grade} \bowtie \text{Professor}))$$

or

$$\text{Student} \bowtie \Pi_{sid}(\sigma_{grade=A}(\text{Grade}) \bowtie \sigma_{name=Hilfinger}(\text{Professor}) \bowtie \text{Student})$$

ii. Relational Calculus

$$\{S \mid S \in \text{Student} \wedge \exists G \in \text{Grade}(S.\text{sid} = G.\text{sid} \wedge G.\text{grade} = A \wedge \exists P \in \text{Professor}(G.\text{pid} = P.\text{pid} \wedge P.\text{pname} = \text{Hilfinger}))\}$$

b) [8 points (4 each)] With your classmates taken care of, it's time to plan your schedule. Write a query that finds the names of the professors who have taught at least one course and have only given grades of "A".

i) Relational Algebra

$$\pi_{\text{pname}}(\pi_{\text{pid,pname}}(\text{Professor} \bowtie \text{Grade}) - \pi_{\text{pid,pname}}(\sigma_{grade \neq A}(\text{Professor} \bowtie \text{Grade})))$$

ii) Relational Calculus

$$\{P1 \mid \exists P \in \text{Professor}(P1.\text{pname} = P.\text{pname} \wedge \exists G \in \text{Grade}(G.\text{pid} = P.\text{pid}) \wedge \forall G \in \text{Grade}((G.\text{pid} = P.\text{pid}) \Rightarrow (G.\text{grade} = A)))\}$$

SID:_____

Question 1 – Formal Query Languages (continued)

c) [7 points] Due to the terrible budget crises at private universities, Berkeley has been able to buy Stanford's entire CS department! It's up to you to integrate Stanford's employee database with ours. Unfortunately, the schema's don't match!

Recall Berkeley's format from before:

Professor(pid, pname, address, department, rating)

Stanford's is similar, except that their professors get paid and the departments are stored separately.

Lecturer(pid, did, pname, address, rating, salary)

Department(did, department)

Write a relational algebra expression that returns the data for the new Professor relation including information for the new professors from Stanford. Exclude Lecturers with a rating lower than 7 (No need to hire every Stanford professor, after all). You can use the relational algebra or the relational calculus, but the algebra will probably be easier.

Note: You should return only those fields that are in the Berkeley "Professor" relation and you can assume that assume "pids" are unique across both the Professor and Lecturer relations (i.e., there are no records with the same pid in both relations).

$$\text{Professor} \cup \pi_{\text{pid, pname, address, department, rating}} (\sigma_{\text{rating} \geq 7} (\text{Lecturer}) \bowtie \text{Department})$$

4) **[2 points]** For question 3 above, why is it important that you be able to assume that pids are unique? (answer briefly):

If pids are not unique, performing the union between the tables could cause a violation of the primary key constraint on pid. (two professors with the same pid)

SID:_____

Question 2 – Buffer Management [5 parts, 30 points total]

a) [15 points] Consider a system with three buffer frames (1, 2, 3) and a file of 9 data pages (A-I). Assume an empty buffer pool and the initial reference counts (i.e., frequencies0 shown below.

Initial Frequency Counts (at T=0)								
A	B	C	D	E	F	G	H	I
11	4	11	2	6	4	4	12	0

A sequence of requests is made to the buffer manager as described in the table below. At certain times a Pin request is immediately followed by an Unpin request (represented as Pin/Unpin), but other times Pin and Unpin requests happen separately.

Starting at time "4", fill in the following table showing buffer contents after the completion of each operation using the **LFU** page replacement policy (as used in HW 1). Unlike in HW1, **increment the reference count for a page only when pinning**. For each page, indicate the pin count (PC) and the resulting reference count (RC) . You can mark unchanged buffer frames with "ditto" (").

Time	**Request**	**Buffer Frames**		
		1	2	3
1	Pin D	D PC=1 RC=3		
2	Pin/Unpin B	"	B PC=0 RC=5	
3	Pin/Unpin C	"	"	C PC=0 RC=12
4	Pin A		A PC=1 RC=12	
5	Unpin D	D PC=0 RC=3		
6	Unpin A		A PC=0 RC=12	
7	Pin H	H PC=1 RC=13		
8	Pin I			I PC=1 RC=1

Question 2 – Buffer Management (continued)

b) [5 points] In HW1 (and the previous question) we never reset the reference counts on pages. That is, references for pages remain part of the reference counts forever. Consider a system that runs for a long time (say, years). Describe a situation where this "never forget" policy could lead to poor performance and say briefly how you might change the policy to avoid this problem.

The first query run by the system touches the same N-1 pages a million times (where N is the size of the buffer pool), then never touches them again. These pages will never get evicted from memory and only the remaining slot will get used.

This could be fixed by periodically decrementing the frequency counts over time.

For parts **c-e** consider a **B+Tree** index that has **64 Leaf pages**. This tree has **7** entries per page (i.e. **8** pointers per page) in the upper levels. The tree is built using Alternative 1 (the actual data records are stored in the leaf pages of the index). Assuming the buffer manager is using LRU page replacement. For the following questions, give a one or two sentence justification for your answer.

c) [3points] With a buffer pool of 2 frames, what page(s) should you expect to be in the buffer? Why?

The just accessed leaf page and it's parent (the index node pointing to that leaf). Every lookup must access the root, then an index page, then a data page, so the root always gets evicted.

d) [3 points] With a buffer pool of 8 frames, what page(s) should you expect to be in the buffer? Why?

The root page and mostly index pages. Each lookup touches three pages, and the root gets touched on every access, so it never gets evicted. Index pages are more likely to be in the buffer since they are accessed more frequently.

e) [4 points] Assume the B+Tree is built on column Y of relation X. Assume that the buffer pool has 32 frames. What hit rate should you expect to get if all the data in the B+Tree is accessed uniformly at random? (less than 25%, between 26 and 50%, between 51% and 75%, or 76%-100%) Why?

76-100%. The buffer pool is large enough to contain the root and the index pages. Each lookup requests the root, an index page, then a data page. The root and index page are going to be hits (in the common case). Then there is a (32-9)/64 % chance the requested data page is also in the buffer pool, giving a (2+(32-9)/64)/3=78% hit rate

SID:_____

Question 3 – Hashing [3 parts, 20 points]:

a) [10 points] Extendible Hashing

Consider the following 5 update operations.

operation no.	operation	key value (binary)
1	insert	24 (011000)
2	insert	46 (101110)
3	insert	20 (010100)
4	insert	26 (011010)
5	Insert	4 (000100)

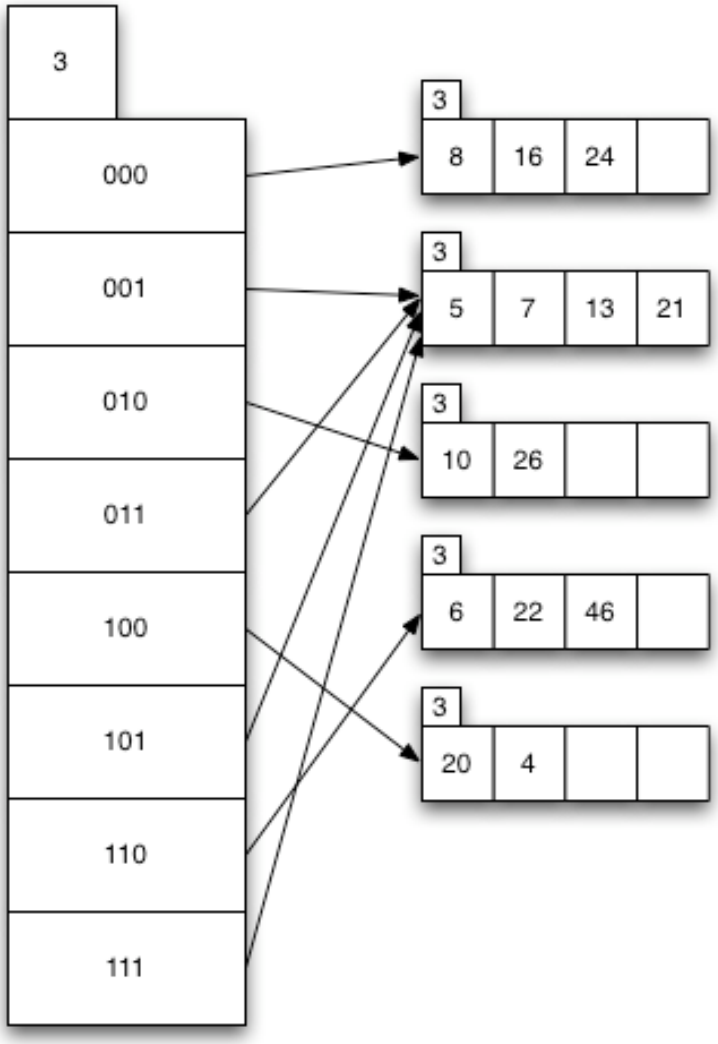
Now, consider an extendible hash structure where each bucket can hold up to 4 entries, with a hash function $h(n) = n \bmod$ and an initial state as shown below.

Draw the extendible hash structure and its contents after the 5 operations have occurred in the order shown. We recommend that you do your scratch work on this page at first. But, this page will not be graded. You MUST put your final answer on the following page!!

SID:_____

Final answer for Question 3(a) - Extendible Hashing:

- Only this page will be graded for question 3(a).
- The final structure should have a directory of size 8 so use the template below.
- Show all buckets and pointers
- Label the directory entries with their corresponding hash value (as on the previous page).
- Make sure to include local depths for all buckets and the global depth of the directory.



Question 3 – Hashing (continued)

b) [7 points] Consider the following index using linear hashing as described in lecture. All search keys are integers. Each bucket can hold at most three (3) search keys. What is the minimal number of search keys you need to insert in order to make bucket "P" in the figure split? Give an example of such a sequence of keys.

Minimum # of Keys: 2

Example Sequence: 32, 11

c) **[3 points]** For the case in part (b), what are the values of "N", "Level", and "Next" after the split of P is performed.

N=4

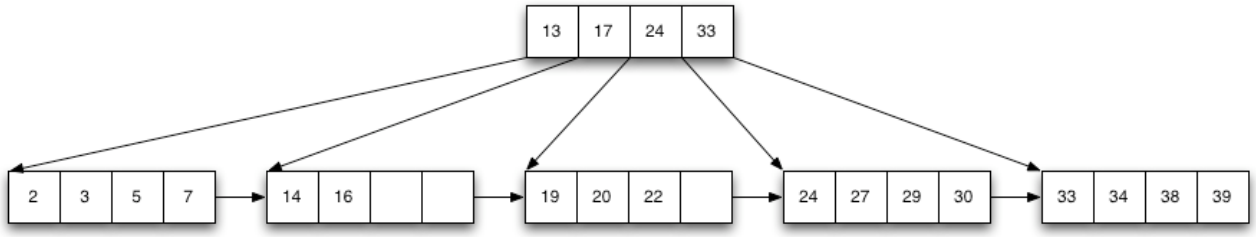
Level=1

Next=0

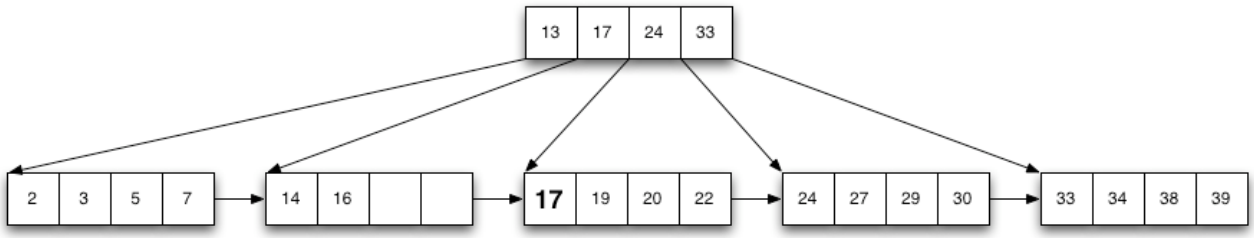
SID:_____

Question 4 – B+Trees [3 parts, 15 points total]:

Consider the following B+Tree with order d = 2.

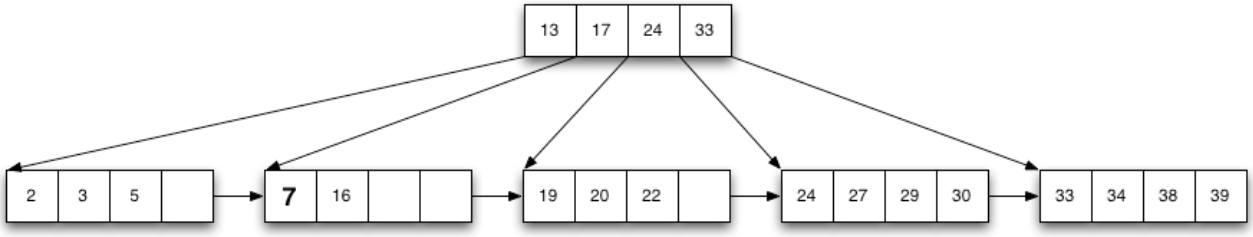


a) [5 points] – Draw the B+Tree that results from inserting the key 17.

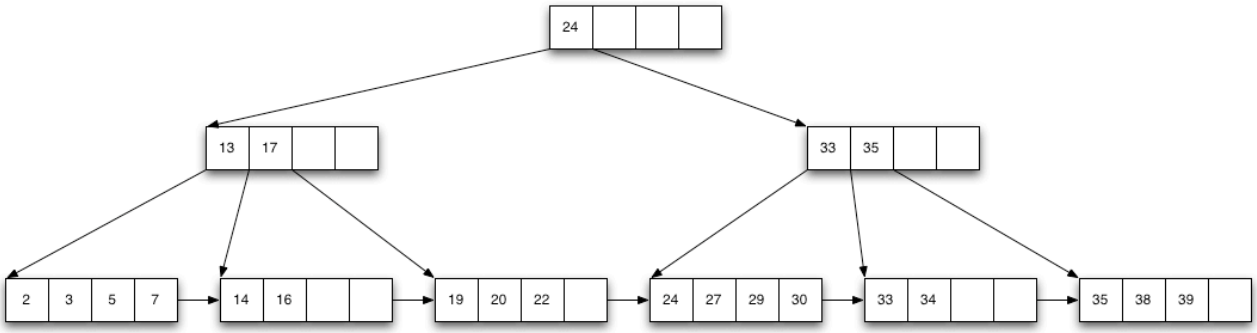


b) [5 points] – **STARTING WITH THE ORIGINAL B+TREE,** Draw the B+Tree that results from deleting the key 14.**Question 4 – B+Trees (continued)**

Recall the original B+Tree with order d = 2 for this question:



c) [5 points] – Again, **STARTING WITH THE ORIGINAL B+TREE,** Draw the B+Tree that results from inserting the key 35.



SID:_____

Question 5 – Short Answer [3 parts, 10 points total]:

For parts **a and b**, indicate what type of index would be best for the following scenarios and explain why. Select from (i) B+ Tree, (ii) static hash, (iii) Extendible Hash, (iv) Linear Hash, or (v) Bulk Loaded ISAM. And briefly explain why.

a) [3 points] A relation where insertions, deletions and range queries are the most common operations.

(i) B+Tree - We need a tree since we want to support range queries. Frequent inserts would cause the creation of overflow pages in an ISAM tree.

b) [3 points] Insertions, deletions and equality selections are the most common operations, and we would like to make access times as uniform as possible.

We want a hash since we don't need to support range queries and trees are slower for insertions, deletions, and equality selections (since we have to navigate down the tree).

(iii) Extendible Hashing – In extendible hashing, we only need to read a single page to find a record. Buckets are immediately split on overflow, so we never have to read an overflow page, like in extendible hashing. (assumes "access"="read")

or

(iv) Linear Hashing – In linear hashing, we only split a single bucket when an overflow occurs - unlike in extendible hashing were we may need to duplicate the (arbitrarily large) directory. (assumes "access"="write")

c) [4 points] Recall that in the slotted page model we use an array of slots containing offsets to point to records inside a page. As described in class, if an existing variable length record grows we just allocate space for it in the free space rather than trying to see if there is room between it and the next record to update it in place. Let's say you wanted to change this. What sequence of steps would you take to not always have to allocate out of the free space when a record grows?

Scan the (entire) slot array to see if there is enough space between the record and its closest neighbor (or if there is enough space between two other neighboring records)